

Using the PowerEditor

The PowerEditor application is used to create complex indicators, studies, systems, and functions, and to modify those that already exist either within TradeStation or those offered by a solution provider.

Some of the tasks you can perform in the PowerEditor can also be accomplished in the QuickEditor utility, which is available with TradeStation. However, the purpose of the QuickEditor is to allow you to create simple analysis techniques, and it therefore contains certain restrictions. The purpose of the PowerEditor, on the other hand, is to enable you to write more powerful and flexible indicators, studies, systems, and functions.

The PowerEditor is simple to use, and users who are familiar with any type of word-processing program will have little difficulty operating in the PowerEditor.

STARTING AND EXITING THE POWEREDITOR

You can start the PowerEditor in two different ways, either directly from Windows or through an Omega Research application such as TradeStation or one of the Omega Servers. This section explains how to access the PowerEditor in both ways, and how to exit the PowerEditor or jump to another application.

Starting the PowerEditor

To access the PowerEditor directly from Windows, follow these directions:

In Windows 95, use the Start - Programs - Omega Research - PowerEditor menu sequence.

In Windows 3.1, go into Program Manager and double-click on the PowerEditor icon, located in the Omega Research group.

To access the PowerEditor through another Omega Research application, click on the GoTo Editor icon located in the Omega applications icon bar.

Exiting the PowerEditor

To exit the PowerEditor, use the File - Exit menu sequence or click on the Close button (X) in the top right corner of the main PowerEditor window.

Jumping to Another Application

You can exit the PowerEditor entirely, or you can jump to another Omega Research application. To jump to TradeStation, use the Tools - GoTo TradeStation option. To jump to an Omega Server, use the Tools - GoTo Server option.

Clicking on either option quickly jumps you to the specific application. If TradeStation is already running, then this choice simply brings it to the foreground. If TradeStation is not running, this choice will run the program and bring it to the foreground. Of course, in order to run TradeStation, you first have to have an Omega Server running, and this applies when jumping to TradeStation from another application.

Note: The GoTo TradeStation and GoTo Server options both have shortcut keys. To jump to TradeStation, press the F11 key. To jump to the Omega Server, press the F10 key.

USING BASIC POWEREDITOR MENU ITEMS

The **File** menu allows you to create new studies, systems, and functions as well as open existing studies, systems, and functions in order to edit them.

The **File** menu also contains menu items that allow the user to print out, delete, protect, verify, and rename analysis techniques. When you start the PowerEditor, some commands in the File menu will be grayed-out. Once either a new file is created or an existing one opened, the grayed-out menu items will become available. This section explains the main PowerEditor commands and menu items as well as the concept of PowerEditor files or windows.

Understanding PowerEditor Windows

When studies, systems, or functions are being created or edited, they are enclosed inside a window. The **Edit** menu is used to make modifications to the active window. The active window can be distinguished from all other open windows because its border is a different color or shade than all others. If there are no open windows, the **Edit** menu is not present.

Many times when modifying existing studies, systems, or functions, certain statements must be found rapidly. The Find options on the Edit menu can be used for just this purpose. It allows users to type in the word or sequence of words for which they are looking. The Replace selection will start from the top of text, searching for the search text, and will replace it with the replace text.

Since more than one study, system or function can be open at once, it becomes important to arrange the windows so they are all visible. This makes it easy to select the window that will be worked on next. The Window menu provides the selections to accomplish this task. Once again, if there are no studies, systems, or functions open, the Window menu is not present. If more than one window is open, the windows can be arranged in the following ways:

OPTION	DESCRIPTION
Arrange All:	Displays all windows in equal-sized panes, with the active window on the left side or upper left corner depending on the number of windows.
Arrange Horizontally:	Displays windows in horizontal, equal-sized panes. The active window appears at the top.
Arrange Vertically:	Displays all windows in vertical equal-sized panes with the active window at the left.
Cascade:	Displays all windows in equal-sized panes in a diagonal overlapping fashion with the active window at the front. The Title Bar for each window is displayed.
Close All:	Closes all open windows
List of Open Windows:	Lists the open windows, with a check mark next to the currently active window. Click on the window you want to make active and bring to the forefront.

Saving Your Work

When writing or modifying a study, system, or function, users will want to be certain to save their work. Each time work is verified successfully, it is automatically saved. Many times, however, an analysis technique may not be ready for verification, yet this work will need to be saved.

The **Save** option will not be displayed until a study, system, or function is edited or created. To save any changes to an existing analysis technique or to save a new analysis technique, use the following directions:

1. Make the desired window active by clicking on its window caption (the window containing the analysis technique to be saved).
2. Use the **File - Save** menu sequence to save the analysis technique.

Important Note: If you edit an analysis technique that was created in the QuickEditor, and/or save it in the PowerEditor, you will no longer be able to edit it using the QuickEditor.

Closing the Active Window

When users are no longer working with a window, they will want to close it to make room for other windows. This task is accomplished with the **Close** option from the **File** menu.

The **File - Close** menu selection will not be displayed until a study, system, or function is opened or created. To close the active study, system, or function, follow these directions:

1. Make the desired analysis technique active by clicking on any part of its window.
2. Use the **File - Close** menu sequence. If the analysis technique has been modified, or has not been verified or saved in the case of newly created analysis techniques, the PowerEditor will ask if the analysis technique is to be saved.
3. Click on **Yes** to save; click on **No** to abort the operation.

The active window will be closed. If any other windows are opened, one of them will become the new active window.

Using the Edit Menu

The items on the **Edit** menu allow you to manipulate the text within the active window of the PowerEditor. Each menu item is described below:

Undo. Selecting the Undo option will reverse all changes made in the last editing operation.

Cut. Highlight the portion of text to be removed (cut) from the active window. Then, use the **Edit - Cut** menu sequence. The text that has been removed can then be pasted into another window.

Copy. The Copy option makes a copy of the highlighted text in the active window. The text that has been copied can then be pasted into another window. To execute the Copy command, use the **Edit - Copy** menu sequence.

Paste. Using the **Edit - Paste** menu sequence will transfer the text stored in the Clipboard (from either the Cut or Copy operation) to the active window on the screen. The location of the transferred text will be determined by the position of the cursor in the active window.

Clear. Highlight the portion of the text to be removed (cleared) from the active window. Then use the **Edit - Clear** menu sequence. The text that has been cleared is not placed in the Windows Clipboard and can therefore not be pasted in to another window or application.

Select All. Choose Select All to highlight all of the text of the active window. Then, if desired, click on one of the other edit options such as Cut or Copy.

Find. Enables you to find a string of text in the active window. You can specify the direction of the search, Up or Down, whether to search for the whole word only and whether or not to make the search case sensitive.

Find Next. Once you have defined the search criteria using the Find option, you can use this option to find the next occurrence of the text string.

Find Previous. Once you have defined the search criteria using the Find option, you can use this option to find the previous occurrence of the text string.

Replace. Enables you to find a string of text in the active window and replace it with another string of text. You can specify the direction of the search, Up or Down, whether to search for the whole word only and whether or not to make the search case sensitive.

CREATING NEW STUDIES, SYSTEMS, AND FUNCTIONS

A **study** is an instruction to TradeStation to plot or draw a mathematical formula in a particular way. An analysis technique creates a visual plot on a chart window. The analysis technique plot may be of various types, such as line, histogram, point, or cross. The information that is plotted and the way in which it is displayed are determined by the user either when the analysis technique is written, edited, or plotted in the charting program. Studies can be indicators, ShowMes or PaintBars.

A **system** is designed to enter and exit the market based on certain rules. A system is an analysis technique that will historically test these rules and advise the user if he or she would have gained or lost financially. In addition, a system will trigger buy, sell, and exit orders to get the user in and out of the market on a tick-by-tick basis. The profit/loss for all open positions will then be monitored, updating every time a new tick comes in.

A **function** provides a shortcut to some of the user's most commonly used mathematical calculations, formulas or values. Instead of typing these calculations, formulas or values continually, you can use a function and type it only once. If the results of the function are to be included in any study, system, or other function, they can be retrieved by using the function name.

The steps involved in creating an indicator, study, system or function are:

1. type in the name of the new indicator, study, system or function
2. type in the text of the indicator, study, system or function
3. verify your work (to make sure the text you have typed is in correct Easy Language format)
4. define its properties

Typing in New Text

To create an indicator, study, system or function, you must first create a new window. Then, once you select the kind of analysis technique to create, a blank window will appear for you to work in. Use the following steps to create an analysis technique:

1. Use the **File - New** menu sequence to access the **New** dialog.
2. Click on the radio button next of the analysis technique you want to create.
3. Click the **OK** button to access the **Create analysis technique dialog**.

Type the name of the analysis technique in the Indicator edit box (or other analysis technique, depending on which type you are creating).

4. Click in the **Notes** edit box, and type in any notes that will help you in identifying this indicator, or other analysis technique, at a glance.

5. Click the **OK** button to produce the **Indicator** window.

6. Type the rules of the analysis technique into the blank window.

7. Use the **File - Verify** menu sequence to verify the newly created analysis technique or press **F3** on your keyboard. Verifying an analysis technique will also save the analysis technique. This command ensures that the text you have typed is in correct Easy Language format.

If no errors are found, the word **Excellent** will appear in the window.

If there is an error, the PowerEditor will identify the type of error found. When the **OK** button is clicked, your cursor will be placed in the exact location of the error.

You must now define the properties, which are set differently depending on the type of analysis techniques with which the user is working.

Defining the Properties of a Study, System, or Function

Using the **File - Properties** menu sequence brings the user to different setting choices depending on whether the active window contains an **indicator, PaintBar, ShowMe, system, or function**.

The **Properties** option will not be visible until an indicator, study, system, or function is opened or created. The choices made with the **Properties** option become the default settings for the analysis technique when it is applied to chart in TradeStation.

Indicator Properties

To set the properties for the indicator that is being edited, or to change the default setting for future indicators, follow the instructions below:

1. Make the window that contains the indicator whose values you want to edit active by clicking on it.

2. Use the **File - Properties** menu sequence to produce the **Indicator Properties** dialog.

3. Click the **Properties** tab to access the **Properties** dialog. The Properties dialog is used to edit the **Number of bars study will reference** setting and to activate or deactivate the study from updating on every tick. It is also used to enable the alerts and the Expert Commentary.

To change the Number of bars study will reference setting, click the radio button of the Auto-detect or User specified option. If you select the User specified option, click in its edit box, delete the current value and type in the replacement value.

Click the Enable Alert radio button to enable a audible and/or visual alert (provided your analysis technique contains the text necessary to trigger an alert).

Click the Update study value every tick radio button to enable the analysis technique to analyze data on a tick-by-tick basis.

4. The Set Default button takes the values just selected and makes them the default settings for any indicators created from that point on. This will not affect any indicators created prior to selecting the Set Default button.

5. Click the OK button to save all changes and return to the PowerEditor main window or go to Step 6 to make changes to the style of the indicator. Click the Cancel button to cancel any changes and return to the PowerEditor main window.

6. Click the Style tab to access the Style dialog. The Style dialog is used to modify the color, style, pen, and width of the plot line(s) of the active indicator or to change the default settings.

7. The Study plot to edit section of the Style dialog lists the plots contained within an indicator. To change a particular plot click the radio button that corresponds to the plot you want to edit. The preview window enables you to see the way the plot displayed on the chart window.

Use the Color, Type, Pen and Width drop down lists to make the desired changes.

8. Click the Set Default button to make the newly selected values the default settings for any indicators created from this point on. This will not affect any indicators created prior to selecting the Set Default button.

9. Click the OK button to save all changes and return to the Omega PowerEditor main window or use Step 10 to make changes to the scaling of the indicator. Click the Cancel button to cancel all changes and return to the Omega PowerEditor main window.

10. Click the Scaling tab to produce the Scaling dialog. The Scaling dialog is used to modify the indicator's scaling. It is critical to scale studies correctly. If the scaling is incorrect, the plot may not be displayed properly or may not even be visible on the screen. Click on the desired option button.

11. Click the Set Default button to make the values just selected the default settings for any indicators created from this point on. This will not affect any indicators created prior to selecting the Set Default button.

12. Click the OK button to save all changes return to the Omega PowerEditor main window. Click the Cancel button to cancel changes and return to the Omega PowerEditor main window.

ShowMe Study Properties

To set the properties for the ShowMe in the active window or to change the default setting for all ShowMe(s) created from this point forward follow the instructions below.

1. Make the window which contains the ShowMe study whose values you want to edit active by clicking on it.

2. Use the File - Properties menu sequence to produce the Properties dialog.

3. Click the Properties tab to produce the Properties dialog. The Properties dialog is used to edit the Number of bars needed to calculate study setting and to activate or deactivate the study from updating on every tick. The Properties dialog is also used to enable Expert Commentary and to enable the alerts.

4. To change the Number of bars study will reference setting, click the radio button of the Auto-detect or User specified option to enable it. If you select the User specified option click in its edit box, delete the current value and type in the replacement value. For more information on the Number of bars study will reference setting, refer to the User's Manual.

Click the Enable Alert radio button to enable a audible and/or visual alert (provided your analysis technique contains the text necessary to trigger an alert).

Click the Update study value every tick radio button to enable the analysis technique to analyze data on a tick-by-tick basis.

The Set Default button takes the values just selected and makes them the default settings for any ShowMe created from this point on. This will not affect any ShowMe studies created prior to selecting the Set Default button.

5. Click the OK button to save all changes and return to the PowerEditor main window or go to Step 6 to make changes to the style of the ShowMe. Click the Cancel button to cancel all changes and return to the PowerEditor main window.

6. Click the Style tab to produce the Style dialog. The Style dialog is used to modify the color, style, pen, and width of the plot line for the active ShowMe or to change the default settings.

The Study plot to edit section of the Style dialog lists the plots contained within a ShowMe study. To change a particular plot click the radio button that corresponds to the plot you want to edit. The preview window enables you to see the way the plot displayed on the chart window.

Use the Color, Type, Pen and Width drop down lists to make the desired changes.

Click the Set Default button to make the newly selected values the default settings for any ShowMe created from this point on. This will not affect any ShowMe studies created prior to selecting the Set Default button.

7. Click the OK button to save all changes and return to the Omega PowerEditor main window. Click the Cancel button to cancel all changes and return to the Omega PowerEditor main window.

PaintBar Study Properties

To set the properties for the PaintBar in the active window, or to change the default setting for all PaintBar(s) created from that point forward, follow the instructions below.

1. Make the window which contains the PaintBar study whose values you want to edit active by clicking on it.

2. Use the File - Properties menu sequence to produce the Properties dialog.

3. Click the Properties tab to produce the Properties dialog. The Properties dialog is used to edit the Number of bars needed to calculate study setting and to activate or deactivate the study from updating on every tick. It is also used to enable the alerts and the Expert Commentary.

To change the Number of bars study will reference setting, click the radio button of the Auto-detect or User specified option to enable it. If you select the User specified option click in its edit box, delete the current value and type in the replacement value.

Click the Enable Alert radio button to enable a audible and/or visual alert (provided your analysis technique contains the text necessary to trigger an alert).

Click the Update study value every tick radio button to enable the analysis technique to analyze data on a tick-by-tick basis.

4. The Set Default button takes the values just selected and makes them the default settings for any PaintBar created from this point on. This will not affect any PaintBar studies created prior to selecting the Set Default button.

5. Click the OK button to save all changes and return to the Omega PowerEditor main window or go to Step 6 to make changes to the style of the PaintBar. Click the Cancel button to cancel all changes and return to the PowerEditor main window.

6. Click the Style tab to produce the Style dialog. The Style dialog is used to modify the color, pen, and width of the plot line for the active PaintBar or to change the default settings

The Study plot to edit section of the Style dialog lists the plots contained within a PaintBar study. To change a particular plot click the radio button that corresponds to the plot you want to edit. The preview window enables you to see the way the plot displayed on the chart window.

Use the Color, Pen and Width drop down lists to make the desired changes.

7. Click the Set Default button to make the newly selected values the default settings for any PaintBar created from this point on. This will not affect any PaintBar studies created prior to selecting the Set Default button.

8. Click the OK button to save all changes and return to the Omega PowerEditor main window. Click the Cancel button to cancel all changes and return to the Omega PowerEditor main window.

System Properties

To set the properties for the system in the active window, or to change the default setting for all system(s) created from that point forward, follow the instructions below.

1. Make the window which contains the system whose values you want to edit active by clicking on it.

2. Use the File - Properties menu to access the System Properties dialog.

3. Click the Properties tab to produce the Properties dialog if it is not already available.

The Properties dialog is used to edit the pyramid settings, entry options, Number of bars needed to calculate study, enable commentary, generate trading orders for the next bar, and set the default settings for the any system created from this point forward.

4. In the Pyramiding sections choose the appropriate setting for whether to allow multiple entries. The choices are Do not allow multiple entries in same direction, Allow multiple entries in same direction by different entry signals and Allow multiple entries in same direction by same and different entry signals

When there are multiple entries in the same direction, the user may scale into or scale out of a position. Consider this example of scaling into and out of a position:

Buy 1 contract
Buy 2 contracts
Buy 2 contracts
ExitLong 2 contracts
Buy 1 contract

In order to close out the position, how many contracts would this user need to ExitLong? A total of 6 contracts have been bought but only two contracts have exited. Four additional contracts would need to exit to close out the position.

Whenever part of a position is exited, TradeStation assumes contracts are exited in the order in which they were put on - First In, First Out.

In the example above, the exit of two contracts would exit one contract bought at trade #1, and one of the two contracts bought at trade #2. Thus, a single position may have multiple trades.

The System Properties dialog allows the user to control scaling into a position. The way in which exit signals are written will determine if they scale out of a position.

If Do not allow multiple entries in same direction is selected, the user will not scale into a position. Once a long or short position is taken, TradeStation will only take exit or reversal orders.

If Allow multiple entries in same direction by different entry signals is selected, the user will only scale into a position if another entry signal other than the entry signal that entered into the current trade is hit.

If Allow multiple entries in same direction by same and different entry signals is selected, the user will scale into a position if either a same entry signal is hit again or if a different entry signal is hit.

Selecting either of the last two options, and depending on the logic of the system rules, may result in X number of buys or entries before a sell or exit is encountered. Thus, a system might buy, buy again, buy a third time, and buy X number of times before selling or exiting even once.

5. In the Entry Option section, specify the number of open entries for each position and the number of contracts or shares for each position. The Maximum Open Entries per position setting refers to the total number of open positions available at any one time. The absolute maximum is 50. The Maximum Entries/Shares per position setting refers to the total number of contracts/shares that are acquired at any one time. The default setting is 65,000.

6. To automate the system, be sure to place a check mark inside the Generate trading orders for next bar check box. TradeStation will then monitor this system on a tick-by-tick basis, generating real-time orders as needed. As the market changes and orders need to be placed or canceled, a signal will occur through both visual and audible alarms. The System Automation

windows (System Open Position, System Active Orders, System Filled Orders, and System Canceled Orders) will display all the activity created by the system.

7. To change Number of bars needed to calculate study setting, click the Auto-detect or the User specified radio button to enable it. If you select the Auto-detect option TradeStation will automatically determine the correct number of bars needed to calculate the study. If you select the User specified option, click in its edit box, delete the current value and type in the replacement value. For more information on the Number of bars needed to calculate study setting, consult the user's manual.

8. Click the OK button to save all changes and return to the Omega PowerEditor main window or use Step 9 to make changes to the system stops. Click the Cancel button to cancel all changes and return to the Omega PowerEditor main window.

9. Click the Stops tab to produce the Stops dialog. The Stops dialog is used to edit the Built-In Stops.

TradeStation has several common built-in stops for the user's convenience. These stops can easily be changed. Users can also write their own exit signals inside their systems. To activate a stop, place a check mark in the check box next to the desired stop. Click inside the corresponding edit box and type in the number that will represent the stop.

10. Click the Set Default button to take these stops and their values and makes them the default settings for any system created from this point on. This will not affect any system created prior to selecting the Set Default button.

11. Click the OK button to save all the changes to the active system, as well as changes to the default values and return to the PowerEditor main window. Click the Cancel button to cancel all changes and return to the PowerEditor main window.

Function Properties

To set the properties of a function in the active window, or to change the default setting for all functions(s) created from that point forward, follow the instructions below.

1. Make the window that contains the function you want to edit active, by clicking on it.
2. Use the File - Properties menu sequence to produce the Function Properties dialog.
3. The Return and Notes edit boxes are provided so that the user can include comments and examples about the purpose of the function for future reference.
4. In the Return Type section click the radio button which corresponds to the correct type of function. The options are Numeric, TrueFalse and String.
5. In the Function Storage section click the radio button which corresponds to the correct type of function. The options are Auto detect, Simple and Series. To have TradeStation make the determination, click on Auto detect. Auto detect is the recommended setting.
6. Click the OK button to save all changes to the active function and return to the Omega PowerEditor main window. Click the Cancel command button to nullify the changes.

Using Functions in Analysis Techniques

To paste a function into the rule being written:

1. Use the Tools - Function Wizard menu sequence to produce the Function Wizard for PowerEditor dialog. A list of the function types is displayed under the Category section.
2. Click on the function type desired. All the functions classified under the selected function type are displayed. Note that at the bottom of the Function Wizard dialog, information is presented relating to the highlighted function. This includes what the function will return, the function inputs, and an example.
3. Click on the function you want to paste in to your analysis technique to highlight it.
4. Click the Paste button.

The function name will be pasted into the active editor window at the cursor position. The function inputs can then be edited to meet the user's requirements. Only one function may be pasted at a time, but as many functions as necessary can be part of a study or system's code.

Finding and Replacing Text in the Active Window

As users write more sophisticated and longer studies, systems, and functions, they will want to be able to find key words or phrases in the text. There will also be instances when there will be one key word, such as a variable name, that needs to be changed throughout the text. This task can be accomplished through the Find and the Find/Replace commands under the Edit menu.

Find. Selecting the Find command will display a dialog that allows the user to enter the word or phrase to be searched for.

The PowerEditor can be configured to Match whole word only (as opposed to a string of text characters) or to look for an exact match of upper and lower case characters. Up or Down may also be selected to determine the direction in which the search will proceed.

Find Next. The Find Next command allows the user to find the next occurrence of the highlighted text. Press the F4 key on the keyboard to accomplish the same task.

Previous. The Previous command allows the user to find the previous occurrence of the highlighted text. Press the F5 key on the keyboard to accomplish the same task.

Replace. The Replace command produces the Replace dialog and allows the user to find a "search string" and selectively replace it with a "replace string." Press the CTRL + R keys on the keyboard to accomplish the same task.

PROTECTING WRITTEN MATERIAL

Select the Protect option to create a password to protect an analysis technique you have written. Remember this password because once the text for the analysis technique is protected with a password, you cannot access it without the password. The Protect menu selection is not visible until a study, system, or function is opened or created. To protect an analysis technique, use the following directions:

1. Make the window that contains the analysis technique active by clicking on it.
2. Use the File - Protect menu sequence to produce the PassWord dialog.
3. A flashing cursor will appear in the Enter Password edit box. Type in a password, then click the OK command button to protect the active analysis technique, or click the Cancel button to terminate the process.

4. When you click OK, the PowerEditor asks you to reenter the password for verification. Type it in, then click OK.

If the password that has been reentered matches the original password, the active analysis technique is protected.

If the password that is reentered does not match, a notification dialog box will appear. Click the OK button to return to the original PassWord dialog, and re-execute Steps 3 and 4.

Once an analysis technique is protected, a padlock will appear next to the analysis technique's name in the title bar.

When an attempt is made to reopen the protected analysis technique, the PowerEditor will ask for the password. If the password is incorrect, the file will not be opened. Notice that when a protected analysis technique is active, the Protect option in the File menu changes to Unprotect. To unprotect a protected analysis technique, make the protected file active and use the File - Unprotect menu sequence. The padlock will be removed and the file will be unprotected. Re-verify the analysis technique to save the change.

MODIFYING AN EXISTING STUDY, SYSTEM OR FUNCTION

Once a study, system, or function has been created, verified and saved, it is added to the appropriate library. In order to view and/or modify a study, system or function, you must first open it. Follow the instructions below to open an existing analysis technique. The instructions are for all analysis techniques.

1. Use the File - Open menu sequence to produce the Open File dialog.
2. Click on the tab of the analysis technique you want to open, for example Indicator. The choices are Indicator, ShowMe, PaintBar, System and Function.
3. Use the scroll bars to locate the analysis technique to be opened.
4. Click the analysis technique or techniques you want to open to highlight them.
5. Click the OK button to open the analysis techniques. Click the Cancel button to terminate the opening process.

A window will appear, containing the rules for the analysis technique you highlighted. If you selected more than one analysis technique to open, more than one window will appear.

Note: Certain functions are read only; you cannot modify them. The functions that are read-only are designated with the letter R.

Copying a Study, System, or Function

Many times, users may want to develop a new analysis technique using much of the Easy Language text found in an existing study, system or function. The Copy and Paste options can be used to copy all or part of the Easy Language text of an existing study, system or function. The copied text can then be pasted into another window and used in a different analysis technique. You can also use the Save As menu choice to save the analysis technique under a new name.

As with the Save option, the Save As option will not be visible until a study, system, or function is opened or created. If a read-only function is active, the Save As option will be grayed out because

read-only functions cannot be modified. Functions that are read-only are designated with the letter R.

To use the Save As option on the active window:

1. Make the window containing the analysis technique active by clicking on it.
2. Use the File - Save As menu sequence to produce the Save Current Study as dialog.
3. Type in the new name.
4. Click the OK button to execute the command. Click the Cancel button to terminate the process.

Once the OK button is selected, the PowerEditor will make a duplicate copy of the active analysis technique under the selected name.

CHANGING THE NAME OF A STUDY, SYSTEM, OR FUNCTION

The Rename button is used to rename an analysis technique or to change the notes on the existing file. Once the file has been renamed, the old name will be replaced with the new name the library of the selected analysis technique.

1. Use the File - Open menu sequence to produce the Open File dialog.
2. Click the tab that corresponds to the type of the analysis technique you want to rename.
3. Click the analysis technique that you want to rename to highlight it. To select more than one analysis technique, hold down the CTRL key as you click on the analysis technique.
4. Click the Rename button to produce the Rename Function (or other analysis technique) dialog.

If you are renaming more than one analysis technique, the Rename Function dialog will be produced once for each analysis technique. Use the No button when you do not want to rename the analysis technique currently displayed in the Rename dialog. The Rename dialog will change to contain the next analysis technique you highlighted.

Type in the new name of the analysis technique.

To make changes to the notes move the cursor into the Notes edit box, delete the current notes and type in the new text.

5. Click the OK button to save changes, or click the Cancel button to abort.

Important: Certain functions are read only; you cannot rename them. The functions that are read-only are designated with the letter R.

PERMANENTLY DELETING AN ANALYSIS TECHNIQUE

When you no longer want to keep a particular study, system or function, you can delete it. Special care should be taken with this command because once an analysis technique has been deleted, it cannot be retrieved.

If one of the analysis techniques that came with the program is accidentally deleted, the only way to retrieve it is to reinstall the program. If a user's analysis technique is accidentally deleted, there is no way to retrieve it.

To delete an analysis technique permanently, follow these directions:

1. Use the File - Open menu sequence. The Open File dialog is displayed with the various tabs: Indicator, ShowMe, PaintBar, System and Function.
2. Click the tab that corresponds to the type of analysis technique you want to delete to make it active.
3. Click the analysis technique you want to delete to highlight it. To select more than one analysis technique hold down the CTRL key as you click on the analysis technique.
4. Click the Delete button. This produces the Delete Confirmation dialog.

If you are deleting more than one analysis technique, the Delete Confirmation dialog will be produced once for each analysis technique.

Use the No button when you do not want to delete the analysis technique currently displayed in the Delete Confirmation dialog. The Delete Confirmation dialog will change to contain the next analysis technique you highlighted.

Use the Yes to All button when you want to delete all the analysis techniques you have highlighted without confirming each one separately.

5. Click OK to delete the analysis technique, or click the Cancel button to abandon the deletion.

Note: Certain functions are read-only; you cannot delete them. When you highlight a read-only function and click the Delete button, you are returned to the PowerEditor and the function is not deleted. The functions that are read-only are designated with the letter R.

FORMATTING THE POWEREDITOR WINDOW

The Editor Options dialog enables you to modify the background color, text color, font and font size used in the PowerEditor. This is also where you change the password that allows you access to the PowerEditor.

Changes are made by scrolling through the selections and highlighting the appropriate choice. All changes are visible via the layout box at the top. Clicking the OK button locks in the changes that have been made.

To access the Editor Options dialog, use the Tools - Options menu sequence.

PRINTING AN ANALYSIS TECHNIQUE

To make a hard copy of an analysis technique including functions, use the Print option.

1. Use the File - Print menu sequence. A secondary menu appears with the choices Active or Other. To print an active analysis technique, click on Active, to print another analysis technique, click on Other.

If **Active** is selected, the **Print** dialog is produced. Skip to Step 4.

OR

If **Other** is selected, the Print Files dialog is displayed with the various analysis techniques:

Indicator, ShowMe, PaintBar, System, and Function.

2. Click the tab which corresponds to the type of analysis technique you want to print.

3. Click the analysis technique or techniques you want to print to highlight them.

4. Click the Print button.

The PowerEditor will print out all analysis techniques that are highlighted.

SHARING AN ANALYSIS TECHNIQUE

The Transfer feature allows the user to transfer studies, systems and functions from one directory to another, or from a hard drive to a floppy diskette. The PowerEditor's transfer feature creates an ELA file to contain the analysis techniques you select for transfer. The ELA file can then be used in another Omega Research application.

When you use the transfer out feature, you have the option of creating a new ELA file to contain the analysis techniques, appending the analysis techniques to an existing ELA file, or overwriting an existing ELA file. Each option is covered in this section.

To transfer analysis techniques out of the PowerEditor, use the following steps:

1. Use the Tools - Open File menu sequence to access the Open File dialog.

2. Click on the appropriate tab to produce the dialog for the type of analysis technique you want to transfer out: Indicator, ShowMe, PaintBar, System, or Function.

3. Highlight the analysis techniques you want to transfer by clicking on them. To select more than one analysis technique, hold down the CTRL key as you click on the analysis technique. You can switch between tabs to select different types of analysis techniques, just make sure you hold down the CTRL key as you click on the analysis technique.

4. When all the analysis techniques you want to transfer are highlighted, click on the Transfer button to produce the Transfer Analysis Techniques dialog.

Make sure the Transfer TO option is selected (its radio button is filled). Since you had analysis techniques selected when you clicked on the Transfer button, the Transfer TO option should already be selected.

5. Click on OK to produce the Transfer to archive file dialog. In this dialog, enter the name of the Easy Language Archive (.ELA) file into which you want to transfer the analysis techniques.

6. In the To edit box, enter the complete path name and file name for the Easy Language Archive (.ELA) file into which the PowerEditor is to transfer the analysis techniques. You can transfer analysis techniques into an ELA file in any directory on your hard drive, a floppy disk or across the network.

To create a new ELA file, use one of the following methods to enter the path name and file name for the new ELA file:

Type in the drive letter, the correct path name (if any), and the name of the ELA file the PowerEditor is to create. For example, A:\FNCTS or C:\TEMP\SYSTEMS or F:\MYDATA\STUDIES, where FUNCTS, SYSTEMS and STUDIES are the ELA file names.

OR

Click on the Browse button to access the standard Windows Save As dialog, in which you can choose the directory and type in the ELA file name the PowerEditor is to create. Once you choose the directory and type in the file name, click on OK. The correct path name and ELA file name are placed in the To edit box in the Transfer to archive file dialog.

To append to or overwrite an existing ELA file, use one of the following methods to enter the path name and file name for the ELA file you want to append to or overwrite:

The PowerEditor remembers where any previously used ELA files are located and includes them as choices in the To drop-down list. Choose an existing ELA file from the To drop-down list.

OR

Click the Scan button to have the PowerEditor scan any drive you specify for existing ELA files. Any ELA files found are placed in the To drop-down list. Choose an existing ELA file from the drop-down list.

6. Once you enter the information in the To edit box using any of the described methods, click on OK.

If you entered the name of a new ELA file, the PowerEditor begins the transfer process and returns you to the QuickEditor when done.

OR

If you entered the name of an existing ELA file, the Overwrite Archive dialog appears. To overwrite the existing ELA file click on the Overwrite button. To add analysis techniques to an existing file, choose the Append button. To return to the Transfer to archive file dialog so you can choose another ELA file, choose **Cancel**.

Once you choose Overwrite or Append, the PowerEditor transfers out the analysis techniques and returns you to the main PowerEditor screen.

The ELA file you created (or appended to) will contain all the files necessary to use the analysis technique(s) with another Omega Research application.

VERIFYING YOUR ANALYSIS TECHNIQUE MANUALLY

The Verify All option is run automatically when you install TradeStation, but you must run the Verify All option manually after you install a new math co-processor or motherboard into your computer. This ensures that all studies, systems, and functions are verified by the PowerEditor. When this option is executed, the PowerEditor checks the grammatical correctness and generates a machine code for all items.

Note: Verify All should not be used for verifying individual analysis techniques. Use the Verify option from the File menu to verify the analysis technique that has been created or modified (you can use the F3 shortcut key instead).

WRITING TRADING IDEAS: A STEP-BY-STEP APPROACH

One of the most powerful aspects of Omega Research software, and TradeStation in particular, is that it provides you with the ability to write your own set of trading rules and test them real-time, without risking a dime. Then, once you test your trading rules and deemed them successful, you can automate your trades and go on to make trading decisions based on your trading system.

In an effort to make the process of writing your own set of trading rules as easy as possible, Omega Research developed a command language that allows you to specify your trading ideas in plain English and at the same time allows the computer to use them. This plain English command language is called Easy Language™. This chapter will teach you how to write your own analysis techniques using the PowerEditor and Easy Language.

There are two general points to look for when converting an idea into Easy Language:

1. What has to happen, that is, what criteria must be met?
2. What action must be taken for what has happened, that is, should a buy or sell order be generated when your criteria has been met?

For example, the criteria that must be met could be that the average of the last 10 bars has to be greater than the average of the last 8 bars. If you wanted to include this criteria in your analysis technique, you can use an existing function that helps you build this idea into your analysis technique. For example, there is a function called Average. The Average function requires that you specify the two numbers that it will use to calculate its final value: a price whose average is to be calculated (such as Open, High, Low, Close or some other mathematical computation) and the length (the number of bars of data to be used in the calculation of the average 20, 50, 100 bars).

Most functions require you to specify the parameters that you want the function to use when it performs its calculations. In many occasions, this is both a price and a length. Since a bar of data is made up of an open, high, low, and close price, the user must specify which particular price the function should use in its calculation. The length refers to the number of bars that the user wants the function to use in its calculation. The "idea" from the above statement has all the information needed to convert it into an Easy Language statement in proper syntax:

If Average(Close,10) is greater than Average(Close,8) then ...

The **Close,10** inside the parentheses specifies that the Average function's value is based on the close price of the last ten bars.

The "What has to happen part" is complete. Now, the "What action must be taken for what has happened" segment must be completed:

Action	Where	Extra Measure	Type Order	Semicolon
Buy	at Open	plus 1 point	stop	;

The semicolon is always to indicate the end of a full statement. The following represents an entire signal as defined by Easy Language:

If Average(Close,10)> Average(Close,8) then buy at close +1 point stop;

At this point, more rules can be added to this system. For instance, the user may add an RSI calculation to this system. For example, the user can add the following rule: buy only if the above rule is true and also if the RSI of the past 14 bars is less than the RSI of the past 20 bars of one day ago. This can be done as follows:

Place the word **"and"** between the two rules. In plain English, the word "and" joins two separate subjects within one principle idea. It is used the same way in Easy Language. Using the example above, add the second "condition" to the overall rule.

If Average(Close,10) > Average(Close,8) and RSI(Close,14) <RSI(Close,20)[1] then

...

As you rules accumulate, you will want to break them down into smaller pieces within true/false variable conditional statements. This makes the rules easier to understand and use. For example:

Condition1 = Average(Close,10) > Average(Close,8);

Easy Language provides you with 100 default conditional variable statement names, Condition0 - Condition99. (Notice there is no space between the word Condition and the reference number.) Now when Condition1 is written in this system, it means:

Average(Close,10) > Average(Close,8);

This Condition1 is only applicable to this system. You may use the variable name Condition1 again in another system or study. The second rule is broken down similarly to the first:

Condition2 = RSI(Close,14) < RSI(Close,20)[1];

RSI is a function already written for the user. Each component is explained below:

RSI(Close,14)	<	RSI(Close,20)	[1];
The RSI function based on the Close of the past 14 bars	Relational Operator: Less than	The RSI function based on the Close of the past 20 bars	Of one bar ago

To write the entire rule, simply plug in the conditions where the text would be:

Condition1 = Average(Close,10) > Average(Close,8);

Condition2 = RSI(Close,14) < RSI(Close,20)[1];

If Condition1 and Condition2 then buy at close + 1 point stop;

This means that if both of the above statements are true, then the system will generate the buy order. If a user wanted a sell order with the same rules but reversed, he or she would write the following:

Condition1 = Average(Close,10) > Average(Close,8);

Condition2 = RSI(Close,14) < RSI(Close,20)[1];

Condition3 = Average(Close,10) < Average(Close,8);

Condition4 = RSI(Close,14) > RSI(Close,20)[1];

If Condition1 and Condition2 then buy at close + 1 point stop;

If Condition3 and Condition4 then sell at close - 1 point stop;

USING INPUTS

Other values can be substituted for Close such as Low, High, Open, Range, Volume, etc. as well as equations. In the event that a user is not certain what values to use for the length of the averages and the RSI, Easy Language uses a feature called an **input**. This feature allows a user to enter one number into the "master" system, but later, when applying and optimizing your systems, it allows the user to change those values without having to change the master code that has already been written.

Assume a user wanted a different length for each of the lengths in the system used in the previous example. The procedure would be as follows.

Inputs: Length1(10), Length2(8), Length3(14), Length4(20), Length5(10), Length6(8), Length7(14), Length8(20);

Inputs: is the inputs declaration that must precede all the different inputs. Each input must have a different name within a system or study, and contain the default value of the input within parentheses to the right of the input name. Each input is then separated with a comma. The inputs are then used in the formulas:

Condition1=Average(Close,Length1)> Average(Close,Length2);

```

Condition2=RSI(Close,Length3) < RSI(Close,Length4)[1]);
Condition3=Average(Close,Length5) < Average(Close,Length6);
Condition4=RSI(Close,Length7) > RSI(Close,Length8)[1];
If Condition1 and Condition2 then buy at open +1 point stop;
If Condition3 and Condition4 then sell at close - 1 point stop;

```

When this system is applied to a chart, it gives the user the ability to substitute different values for each length, which allows optimization of all values against each other.

USING VALUES

Another possible substitution is to set a particular number to a **value variable**. Value variables are like conditional variables, in that they store the results of a calculation. However, where the results stored in a conditional variable are true or false only, the results stored in a value variable are **numeric** only. Easy Language provides 100 default value variable names, Value0 - Value99. (Notice that there is no space between the name and the reference number.)

In the following example, "1 point" is assigned to the value variable **Value1**. The ability to assign values to value variables saves the user time and effort, because rather than having to retype the value continually, he or she simply assigns the value to the value variable once at the beginning of the Easy Language text and then uses the value variable throughout the text.

```

Inputs: Length1(10), Length2(8), Length3(14), Length4(20), Length5(10),
Length6(8), Length7(14), Length8(20);
Value1 = 1 point;
Condition1 = Average(Close,Length1) > Average(Close,Length2);
Condition2 = RSI(Close,Length3) < RSI(Close,Length4)[1]);
Condition3 = Average(Close,Length5) < Average(Close,Length6);
Condition4 = RSI(Close,Length7) > RSI(Close,Length8)[1];
If Condition1 and Condition2 then buy at open + Value1 stop;
If Condition3 and Condition4 then sell at close - Value1 stop;

```

The following are a few more examples of the use of conditional and value variables. Assume the user wants the system to place an order, either on the long or short side if it's Monday or Friday, and this order is to be placed on the following day's open.

```

Inputs: Length1(10), Length2(8), Length3(14), Length4(20), Length5(10),
Length6(8), Length7(14), Length8(20);
Value1=1 point;
Condition1=Average(Close,Length1)> Average(Close,Length2);
Condition2=RSI(Close,Length3) < RSI(Close,Length4)[1]);
Condition3=Average(Close,Length5) < Average(Close,Length6);
Condition4=RSI(Close,Length7) > RSI(Close,Length8)[1];
If Condition1 and Condition2 and DayofWeek(Date) =1 or DayofWeek(Date)=5 then
buy at open + Value1 stop;
If Condition3 and Condition4 and DayofWeek(Date) =1 or DayofWeek(Date)=5 then
sell at close - Value1 stop;

```

All of these instructions can be written as one system with "and" between each separate rule.

TESTING NEW IDEAS

Many times traders want to enter the market only if certain prices or studies have crossed certain levels 2, 3, 4 or many more times. Easy Language allows this to be done in very simply through a device called a counter.

Assume a user wants to buy when the RSI has crossed over an 80 buy zone line 5 times during a 25 bar period. In this case, the program will need to be able to evaluate how many times the RSI has crossed that line.

To do this, the user would use a counter, as shown in the following example.

```

If RSI(Close,25) crosses above 80 then Value1 = Value1 + 1;
If Value1=5 then begin
    Buy tomorrow at open + 1 Point stop;
    Value1=0;
end;

```

Broken down, the above signal means the following:

RSI(Close,25) ... the RSI value based on the closes of the past 25 bars.

crosses above ... indicates that a value has "**crossed above**" a value, in this case the buy zone line.

80 ... buy zone line value.

Value1 = Value1 + 1 ... Value1 is the counter, and in this example, is being incremented by 1 each time the RSI crosses above the 80 buy zone line. Once Value1 reaches 5, the system will perform the specified action, which in this case, is generate a buy order.

If Value1 = 5 then buy tomorrow at open + 1 point stop; ... once Value1 = 5, the program will place a buy order on the open of tomorrow and fill it if the price rises one point above the open.

Value1 = 0; ... after Value1 = 5 and the action has taken place, Value1 is reset to zero.

Testing Another Idea

Assume a user wants to place a buy order when a 10-day RSI crosses above a 14-day Stochastic 3 times, and to place a sell order when the 10-day RSI crosses below the 14-day Stochastic 4 times.

To do so, the user can write the following Easy Language statements:

Condition1 = RSI(Close, 10) crosses above SlowD(14);

Condition2 = RSI(Close,10) crosses below SlowD(14);

If Condition1 then Value1 = Value1 + 1;

If Value1 = 3 then begin

Buy today at close;

Value1 = 0;

end;

If Condition2 then Value2 = Value2 + 1;

If Value2 = 4 then begin

Sell today at close;

Value2 = 0;

end;

Broken down, these instructions mean the following:

Condition1 = RSI(Close, 10) crosses above SlowD(14);

Condition2 = RSI(Close,10) crosses below SlowD(14);

Writing such rules as conditional variables makes writing the system considerably easier and cleaner. When the RSI crosses above the SlowD, Condition1 = True; and when the RSI crosses below the SlowD, Condition2 = True.

If Condition1 then Value1 = Value1 + 1;

If Value1 = 3 then begin

Buy today at close;

Value1 = 0;

end;

Each time the result of Condition1 is True, in this case Value1, increments by 1; and each time the result of Condition2 = True, the counter, in this case Value2, also increments by 1. Once either counter reaches 3, the specified action is performed. Once the action is performed, the counter is reset to zero.

If Condition2 then Value2 = Value2 + 1;

If Value2 = 4 then begin

Sell today at close;

Value2 = 0;

end;

Value variables are useful tools when writing different types of systems and/or studies. Below are additional examples of value variables.

Value1 = 2 * H;

Value2 = Average(RSI(Close,14),10) - Average(RSI(Close,10),10);

Value3 = 50 points;

Value4 = High - Low + (Range[1] *.5);

Keep in mind that value variables can only store numeric expressions, that is, expressions that result in a numeric value. They cannot store True/False expressions.

USING FUNCTIONS TO HELP WITH EASY LANGUAGE

As trading ideas written in Easy Language, the user will begin to notice that many of the desired concepts are already "built-in" to the program. These concepts usually perform a particular function within the Easy Language system or indicator. For instance, there is a function that looks for the average of a price. The user need only put it into the Easy Language structure the program can understand.

The following are a few examples that can helpful in writing trading ideas:

Taking the average of the last 10 closes:

Average(Close,10)

Where:

Average...the function that returns the value requested.

(Close,10)...the price and length of the data on which the average is to be calculated.

Getting the results for an RSI over the last 20 Opens:

RSI(Open,20)

Where:

RSI...the function that returns the value requested.

(Open,20)...the price and length of the data on which the RSI is to be calculated.

While writing systems or studies, the user has the ability to include or paste in functions. The user then enters the inputs he or she wants the function to use to perform its calculation. Some examples of functions and their possible inputs are listed next:

Average(Close,14) RSI(High,20) ADX(20) Highest(Close,15)

StdDev(Open,12) WAverage(Low,10) SwingHigh(2,High,3,20)

Six of the above functions simply require a price and/or length (number of bars) on which to perform their calculations. However, the last example, SwingHigh, requires that more values be entered:

SwingHigh(Occur,Price,Strength,Length)

A **SwingHigh** represents a point on a chart that is higher than the points to its right and at least as high as the points to its left.

Occurrence is the immediacy of the SwingHigh. In other words, whether it's the most recent occurrence, one back, second one back, third one back, etc.

Price refers to the portion of the bar that is of interest, for example, the high, low, close, open price, etc. and which function will use in its calculation.

Strength is the number of points (bars to the left and to the right) that the SwingHigh point must be higher than.

Length is the number of bars used in the SwingHigh calculation.

WORKING WITH STUDIES

Many of the previous examples in this online manual have dealt with systems. However, the same rules apply when writing studies (indicators, PaintBars and ShowMes). The major difference is that studies plot information, whereas systems actually place buy and sell orders based on the rules the user has written.

Following is an example of a system we used previously:

Inputs: Length1(10), Length2(8), Length3(14), Length4(20), Length5(10), Length6(8), Length7(14), Length8(20);

Value1 = 1 point;

Condition1 = Average(Close,Length1) < Average(Close,Length2);

Condition2 = RSI(Close,Length3) > RSI(Close,Length4)[1];

Condition3 = Average(Close,Length5) > Average(Close,Length6);

Condition4 = RSI(Close,Length7) < RSI(Close,Length8)[1];

Assume the user wants to change this into an indicator. First, break up the conditional statements into value statements, as follows:

Value1 = Average(Close,Length1);

Value2 = Average(Close,Length2);

Value3 = RSI(Close,Length3);

Value4 = RSI(Close,Length4);

Assume that for the indicator, these are the only values in which the user is interested. The Plot statements would be written as follows:

Plot1(Value1, "Avg1");

Plot2(Value2, "Avg2");

Plot3(Value3, "RSI1");

Plot4(Value4, "RSI2");

Where:

PlotN ... is the Plot declaration statement.

ValueN ... is the result of that calculation.

AvgN ... flags the name of the plot for viewing in charting.

Plot statement inputs are surrounded by parenthesis and separated by commas.

The following is an example of another indicator: assume a user wants to plot the highest high of the last 20 bars and the lowest low of the last 10 bars. He or she also wants to plot the 10-bar moving average subtracted from the 14-bar moving average of 2 bars ago.

The first rule is....plot the highest high of the last 20 bars:

Question 1. Is there a function for Highest?

Answer: Yes.

Question 2. What price is wanted to calculate the Highest function?

Answer: High.

Question 3. For what period of time?

Answer: 20.

This study can be written in either of two ways. The first is simply to write plot statements with the rules within the statement; the second is to assign each of these separate calculations to a value variable. Examples of both options follow:

If the rule were assigned within a plot statement, this would be the result:

Plot1(Highest(High,20), "Highest");

If the same rule were assigned to a value variable, this would be the result:

Value1=Highest(High,20);

Plot1(Value1, "Highest");

The second rule of the idea is...reveal the lowest low of the last 10 days. Using the same questions as in rule one, the answers are: yes, there is a function for Lowest; the price being sought is "low;" and the period being sought is 10 bars.

Assignment within a Plot statement:

Plot2(Lowest(Low,10), "Lowest");

Assignment to a Value:

Value1=Lowest(Low,10);

Plot2(Value1, "Lowest");

The final rule within the idea is to plot a 10-day moving average subtracted from a 14-day moving average starting 2 days ago. The same process can be used as was used above; find the function, the price on which to calculate and the length of each. When this calculation is assigned to a value, it appears as follows:

Value3=Average(Close,14)[2] - Average(Close,10);

Plot3(Value3, "Average");

Review the above example carefully. Remember that value variables are numeric variables. The above equation, Average(Close,14)[2] - Average(Close,10), produces a number. If instead of using a "mathematical operator," a "relational operator" such as > or < were used, then this equation should be assigned to a condition variable, which stores a true/false expression.

This equation can also be written as a plot statement:

Plot3(Average(Close,14)[2] - Average(Close,10), "Average");

Now put all the rules together. First the study with the values assigned:

Value1=Highest(High,20);

Value2=Lowest(Low,10);

Value3=Average(Close,14)[2] - Average(Close,10);

Plot1(Value1, "Highest");

Plot2(Value2, "Lowest");

Plot3(Value3, "Average");

Next the study with the Plot statements only:

Plot1(Highest(High,20), "Highest");

Plot2(Lowest(Low,10), "Lowest");

Plot3(Average(Close,14)[2] - Average(Close,10), "Average");

The Highest and Lowest functions simply find the highest and lowest values and plot them. Since the two main pieces of the indicator use different scales. Therefore, the solution is to split the one indicator into two different indicators, as shown:

First Indicator

Value1=Highest(High,20);

Average(Close,10);

Value2=Lowest(Low,10);

Value3=Average(Close,14)[2];

Value4=Average(Close,10);

Plot1(Value1, "Highest");

Plot2(Value2, "Lowest");

Plot3(Value3, "Average");

Plot4(Value4, "Average2");

Second Indicator

Value1=Average(Close,14)[2]-

Plot1(Value1, "Average3");

The same type of logic used for writing indicators can be used for writing ShowMe and PaintBar studies. It is most important to remember that an indicator, ShowMe or PaintBar plots information. They plot numbers or numeric results of an equation, etc. They can be used not only as visual aids but they can also be written such that they notify (or alert) you when certain market conditions occur.

DEFINING ALERTS WITHIN STUDIES

One of the best features of TradeStation is the ability to write alerts within studies. For example, using the indicator from the previous section, write an alert for it.

```
Value1=Highest(High,20);
Value2=Lowest(Low,10);
Value3=Average(Close,14)[2];
Value4=Average(Close,10);
Plot1(Value1, "Highest");
Plot2(Value2, "Lowest");
Plot3(Value3, "Average1");
Plot4(Value4, "Average2");
```

One of the words the user may start using after working with Easy Language for a while is the word "flag."

Flag refers to signaling that a particular action needs to take place. In the case of alerts, the user must "flag" the indicator that another action needs to be taken. To do this, the program uses the reserve word, CheckAlert. In the Easy Language structure, the IF/THEN syntax is then used:

If CheckAlert then ...

This simply means that if the user wants to check for an alert, begin doing so on the following.

The only item TradeStation needs now is the criteria for the alert. Assume that a user wants to be alerted if **Plot3** crosses above **Plot4**. If the value of that plot crosses above the other, TradeStation will alert the user to that fact. An equivalent rule is "or **Plot4** crosses below **Plot3**."

If Plot3 crosses above Plot4 or Plot4 crosses below Plot3

After specifying the rules, which is the "IF" portion of the statement, you have to specify the "THEN" portion, which is the action that needs to be taken. So at this point, the alert text is as follows:

If CheckAlert then if Plot3 crosses above Plot4 or Plot4 crosses below Plot3 then ...

What should happen if the criteria are met? TradeStation must alert the user. So, after "Then" write the word "Alert." If either of those rules is true, the alert will be met; therefore, the alert = true. Write it that way in the alert text:

If CheckAlert then if Plot3 crosses above Plot4 or Plot4 crosses below Plot3 then Alert = True;

Therefore, the entire indicator with alert would appear as:

```
Value1=Highest(High,20);
Value2=Lowest(Low,10);
Value3=Average(Close,14)[2];
Value4=Average(Close,10);
Plot1(Value1, "Highest");
Plot2(Value2, "Lowest");
Plot3(Value3, "Average1");
Plot4(Value4, "Average2");
If CheckAlert then if Plot3 crosses above Plot4 or Plot4 crosses below Plot3
Then Alert = True;
```

Once the alert has been triggered by the criteria that has been written, the **Attention** dialog will appear, advising of an alert along with an audible tone.

WRITING A FUNCTION

The first step in writing a function is deciding on the name for the new function. This name is what will be used in any study or system that references the function. It should therefore be a name that will be as representative as possible of the action the function is to take (the function name must be less than 20 characters in length and contain no spaces).

The second step is the function itself. Assume a user wishes to write a function that takes a 10-bar average of a 14-bar RSI and adds that to the bar's Range * 0.5.

Name	Assignment	Equation
MYRSICALC	=	Average(RSI(Close,14),10)+(RANGE * 0.5);

That's all there is to it. Now, any time the user wishes to use this formula in one of his or her studies or systems, all they have to do is include this function anywhere in the system/study where that calculation is to be performed. You can change the above example so that inputs are called instead of one number every time. The following would be the result:

Inputs: Price1(NumericSeries), Length1(Numeric), Length2(Numeric);

<u>Name</u>	<u>Assignment</u>	<u>Equation</u>
MYRSICALC	=	Average(RSI(Price1,Length1),Length2)+ (Range*0.5);

As can be seen, there is one major difference between this input and the inputs in studies or systems. In functions, the user needs to specify the category of the input, not the default value of the input. In other words, specify the type of input it is, whether Numeric(20), NumericSimple (price with no length attached), NumericSeries (price with length attached), or the TrueFalse category. As long as the user understands which category the function falls into, there should be no problem in assigning it.

UNDERSTANDING EASY LANGUAGE

In 1987, Omega Research developed Easy Language, a plain English command language that provides users with no programming knowledge the flexibility and power to take their own ideas and turn them into studies or systems that could be applied within TradeStation. Easy Language has incorporated many of the terms used in stock and commodity. Ideas, when written in Easy Language, are then translated into a machine language that can be recognized by the computer. This allows users to write their ideas just as they would describe them to another trader.

There are three basic types of analysis techniques that can be constructed using Easy Language: Studies, Systems, and Functions. This chapter will serve as a reference, covering all the elements of the language.

Studies perform a calculation based on a certain set of rules written using Easy Language and plot the results on the screen. These plots can then be compared to the data or to one another if more than one plot is used. There are three study types: Indicator, ShowMe, and PaintBar Studies, and they are all very similar in the way they are constructed.

Systems also have a set of rules written using Easy Language, but they perform a different function. Systems place buy, sell, and exit orders on the computer screen when the rules of a user's System have been met. When an order is placed, a position is opened. That position can then be monitored for profit and loss information. This feature is only available with systems.

Functions can be thought of as a shortcut list. If a user has a certain set of rules that he or she would like to use often, these can be written inside a function. That function can then be called, by using a single word, inside any study, system, or even another function. This saves time and also makes the other analysis techniques easier to understand.

Like any language, Easy Language has a set of basic components that make up the core of the language. Components are words, characters, or methods used to construct statements. These components must be organized and structured correctly to create valid sentences or statements. The organized pattern or structure of language is called syntax. If a statement does not have correct syntax, the computer will be unable to recognize it as valid. When the components of Easy Language are used in the proper syntax, virtually any trading strategy can be written.

One of the major components of Easy Language is the function. Many functions were written by Omega Research, Inc. for the convenience of the user; however, the user may write his or her own functions. Then, all that users have to do is to reference the function while writing their own studies, systems, or other functions. Functions that are already included with TradeStation are covered in further detail later in this chapter.

Ultimately, the goal will be to write complete statements using Easy Language. We will first look at how to put the different components of the language together to form various expressions. Then we will discuss how to turn these expressions into statements. Once a user is able to write statements, he or she will have little difficulty turning ideas into studies or systems that TradeStation can read and recognize.

The following topics:

Turning Components Into Expressions

This section will focus on the glue that holds together the individual components used to make a valid expression. Each component of the language will be covered later in this chapter.

The ability to compare one item with another is important because it turns price, values, etc., into Easy Language expressions. It lets users compare today's open with yesterday's open, or today's volume with yesterday's volume, or the low of the current bar with the low of the last 10 bars. Taking it one step further, a user may want to know if today's open is greater than yesterday's open, or if the low of the current bar has been the lowest low for the last 10 bars.

Easy Language can perform simple mathematical calculations on prices or values. It can also make both multiple comparisons and mutually exclusive comparisons. For example, a user may want to know if the open and close of the current bar are greater than the open and close of the previous bar. In another example, a user might want to know if the open or close of the current bar is greater than the open or close of the previous bar.

The elements needed to perform all these functions are called operators. An operator is a connection between words or numbers. Operators may themselves be either words (and, or) or symbols (+ - * / > <). We call these connectors "operators" because they refer either to a mathematical, relational, or logical operation.

Performing Basic Math

The mathematical operators (+ - * / > <) perform the four basic mathematical functions of multiplication, addition, subtraction, and division. No other words or special characters can be used

to perform these operations. Mathematical operators provide an important tool to work with when constructing Easy Language expressions.

For example, to determine the range of a bar, the mathematical operator (-) would be used, as follows:

High - Low

There are four available mathematical operators:

OPERATOR MEANING

+	Addition
-	Subtraction
*	Multiplication
/	Division

Comparing Prices, Values, and Other Elements

The special characters necessary to make a comparison between one item and another are greater than, less than, equal to, greater than or equal to, less than or equal to, and not equal to. TradeStation adds two other operators: crosses over and crosses under to this list. These two operators are extremely convenient shortcuts that are beneficial when trying to identify a cross over or cross under between two separate items.

These special characters include:

<u>OPERATOR</u>	<u>MEANING</u>
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
=	Equal to
<>	Not equal to
crosses over/above	Greater than on current bar but less than or equal to on the previous bar
crosses under/below	Less than on current bar but greater than or equal to on the previous bar

These special characters typically describe the relationship between prices. Crosses over/above and crosses under/below depend on the previous day's condition not being true.

<u>EXPRESSIONS</u>	<u>DESCRIPTION</u>
H < H[1] + 1 point	High of current bar less than high of previous bar plus one point
L[0] <= L[1]	Low of current bar less than or equal to low of previous bar
C > C[3]	Close of current bar greater than close of three bars ago
C >= Close[3]	Close of current bar greater than or equal to close of three bars ago
H <> H[1]	High of current bar not equal to high of previous bar
O crosses above C[1]	Open of current bar crosses above the close of the previous bar

Combining Two or More Expressions

There are two special words used to link two or more comparisons.

And

Or

The **And** combines a series of relational-type expressions into a compound expression. That compound expression is more restrictive (i.e., more difficult to achieve a **true** result) than the stand-alone expressions on either side of the **And**. When using **And**, the expressions on either side of the word must be **true** in order for the entire expression to be evaluated as **true**; however, if either expression is **false**, then the entire expression is evaluated as **false**.

In the example below, a user wants to identify when volume is within a certain range. The volume would have to be greater than 13,000 and less than 23,000 for the expression to be evaluated as **true**. If either of the two conditions is **false**, however, the entire compound expression is evaluated as **false**.

Volume > 13000 and Volume < 23000

The **Or** also combines a series of relational-type expressions into a compound expression. However, unlike **And**, **Or** makes the compound expression less restrictive (i.e., easier to achieve a **true** result) than the stand-alone expressions on either side of the **Or**. When using **Or**, if either stand-alone expression is evaluated as true, then the entire compound expression is evaluated as **true**. For the compound expression to be evaluated as **false**, both stand-alone expressions must be evaluated as **false**.

In the example below, if the close of the current bar was greater than the close of the previous bar, or the open of the current bar was less than the open of the previous bar, the compound expression would be evaluated as true. However, if both stand-alone expressions were false the compound expression would be false.

Close > Close[1] or Open < Open[1]

Are Expressions Being Evaluated Correctly?

When several operations take place in a numeric expression, they are evaluated according to a well-defined hierarchy so that the results are always predictable.

When evaluating statements, the following order of precedence is used. If two or more operations are of the same level of precedence, operations are executed left to right.

Parentheses

Multiplication and division

Addition and subtraction

<, >, =, <=, >=, <>

And

Or

The following guidelines and examples help to illustrate the order of precedence.

The order can be changed by using parentheses. Operations within parentheses are performed first. The usual order of operations is maintained inside the parentheses. For example, the expression $8 - (3 - 2)$ is evaluated as: $3 - 2 = 1$, then $8 - 1 = 7$. The expression

If the close of today - (close of 3 bars ago - open of 3 bars ago)...

... is evaluated as:

Close of 3 bars ago - open of 3 bars ago = X; If the close[0] - X...

With nested parentheses, evaluation moves from the innermost level to the outermost level. For example, the expression

$4 * (2 * (3+4))...$

... is evaluated as:

$3 + 4 = 7$; $2 * 7 = 14$; $4 * 14 = 56$.

The expression

Close of today + $0.75 * (\text{close of 3 bars ago} - \text{close of 2 bars ago})...$

... is evaluated as:

Close of 3 bars ago - close of 2 bars ago = value X

Then $0.75 * X = \text{value Y}$

Finally, close[0] + Y yields result Z.

A Review:

What is the product of $4 + 5 * 6$? Is it $4 + 5 = 9$; then $9 * 6 = 54$?

This is incorrect. Remember that multiplication takes precedence over addition. Therefore, the product is calculated:

$5 * 6 = 30$; then $30 + 4 = 34$.

Can these numbers be written in such a way that, using the same operators, the product is 54? Of course. Simply add parentheses. $(4+5) * 6 = 54$.

Finally, assuming that today's open was 250.00 and yesterday's close was 240.00, is there any difference between statement 1 and statement 2?

1. Open of today + Close of yesterday /2

2. (Open of today + Close of yesterday)/2

Remember, division takes precedence over addition. So statement 1 reads:
 $250.00 + 120.00 = 370.00$

However, parentheses take precedence even over division. Statement 2 reads: $490.00 / 2 = 245.00$
There are a lot of dollars between 245 and 370.

Parentheses are used to separate specific items that have a relationship to one another that supersedes or is superior to the relationship they have to the rest of the expression.

For example, consider the following expression:

$X = A + B * C$

In mathematics, multiplication has a higher precedence than addition. So, in order to find a value for **X**, multiply **B** times **C**, then add that result to **A**. The solution is **X**.

However, in the following example, parentheses change the order of operations.

$X = (A + B) * C$

In this case, first add A plus B, and then multiply that sum by C. Is the answer in both cases the same? Assign the following values: $A = 1$; $B = 2$; $C = 3$.

In the first example, $2 * 3 = 6$, and $1 + 6 = 7$. In the second example, $1 + 2 = 3$, and $3 * 3 = 9$.

The two answers are not alike because parentheses have changed the order of precedence, or the order in which operations are performed. Precedence affects mathematical operators, relational operators, and even logical operators.

In order to prevent confusion, in TradeStation may not contain both the words And and Or unless they are separated by parentheses. In the following example, both statements have correct syntax; however, they return different results.

(4 > 5 and 3 > 7) or 1 < 3 True statement
4 > 5 and (3 > 7 or 1 < 3) False statement

Long statements containing And and Or operators can be difficult to understand and debug. Use the Logic Tables below as a guide.

CONDITION1	OPERATOR	CONDITION2	OVERALL
TRUE	AND	TRUE	TRUE
TRUE	AND	FALSE	FALSE
FALSE	AND	TRUE	FALSE
FALSE	AND	FALSE	FALSE
TRUE	OR	TRUE	TRUE
TRUE	OR	FALSE	TRUE
FALSE	OR	TRUE	TRUE
FALSE	OR	FALSE	FALSE

Thus, the use of **And** lessens the chances that a true outcome will occur because both conditions (or all conditions in the case of multiple **And** operators) must be true for the result to occur:

5 < 10 and 60 > 50

The use of **Or**, however, increases the likelihood that a true result will occur because only one condition must be true:

5 < 10 or 60 > 50

Remember that in order to prevent confusion, in TradeStation, a statement may not contain both the words **And** and **Or** unless they are separated by parentheses.

Taking Expressions and Making Them Into Statements

Once a user learns to construct expressions, the next task is to take those expressions and turn them into statements. What is a statement? A statement is an expression or series of expressions **put into the correct syntax**.

Note: Statements must end with a semicolon.

There are a number of ways to write statements. One type often used is called the simple IF - THEN statement. The IF section tells what must happen before the THEN section may be executed. Logically, then, the IF segment must always come before the THEN segment.

The IF section of the statement is made up of one or more expressions. For example:

Close > Close[1] and Close[1] > Close[2]

The THEN section of the statement tells the action to take. For example:

Buy at Open

Now drop the IF - THEN and the semicolon into it to form the complete statement, as shown in the figure below.

Example of a simple If-Then statement

This is the long entry portion of a system that takes a long position when there have been two consecutive higher closes.

Types of Statements

The statements used in a study or system instruct TradeStation to take certain actions. An expression may be made up of a number of different words, numbers, or operators. One or more expressions constructed in the correct syntax in turn form a statement. A Study or System is made up of one or more statements.

There are a number of different statements: PLOT, PRINT, BUY, SELL, EXIT, ASSIGN, INPUT, VARIABLE, ARRAY, LOOP, SIMPLE IF-THEN, BLOCK IF, and NESTED IF statements. All

statements, however, have one thing in common: they all end in a semicolon, which signifies the end of a statement.

The construction of the statements displayed in the following section are only applicable in the PowerEditor. The standard Editor has its own set of rules for the creation of studies, systems, and functions.

What Are Variables and How Are They Used?

A variable, as defined by Webster, is **that which is subject to change; a symbol that may have an infinite number of values**. A variable is used as a storage location to hold the result of some numeric or logical calculation. The benefit of using a variable is that once it is assigned it can be called without having to reiterate its assignment. In Easy Language, there are two types of variables: numeric and logical. Numeric variables store numbers while logical variables store one of two values, **True** or **False**.

For example, variable **X** can be assigned to equal the Close of the bar minus 10% of the bar's range. This variable **X** could then be used numerous times throughout the Study without having to retype the formula.

Declaring a Variable

Before variables can be assigned and used, they must be declared. This is done in a Variable Declaration statement. The assignment of the variable takes place in an Assignment statement. Variable names should be descriptive of their content. For example, if the variable contains an average of the last two closes, it might be named TwoClose. Variable names cannot exceed 20 characters in length, cannot use spaces, and cannot use special characters (such as the characters above the numbers keys on a keyboard). Variable names can have numbers in them; however, the number or numbers cannot start the variable name. This means that all variable names must start with a letter. In addition, variable names cannot be one of the reserved words listed in the section Reserved Words.

The program was sent with 100 pre-declared numeric variables and 100 pre-declared logical variables. Numeric variables called Value0 through Value99 have been initialized to zero, whereas logical variables called Condition0 through Condition99 have been initialized to false. If you decide to use these variables, there is no need to re-declare them. The drawback to using these variables is that they are not very descriptive.

The Variable Declaration statement contains the name of the variable, the default or starting value within parenthesis, and a semicolon. More than one variable can be declared by inserting a comma between variables. The word Variables, Var, or Vars can all be substituted for the word Variable in the syntax. The word VariableName would be replaced by a valid variable name. The words Default Value would be replaced with a number, usually 0, in the case of numeric variables. In the case of logical variables, the word Default Value would be replaced with either the word False or True. This is done as a precaution to ensure that on the first bar, the variable does not have the incorrect starting value. In most cases, it does not make sense to set the default value to anything else but False. The semicolon is used to indicate the end of the statement.

Variable Declaration statements should be placed at the top of the code and must be placed before the Variables are assigned.

Variable Declaration Syntax: Variable: VariableName(Default Value);

The first figure below demonstrates a numeric Variable Declaration and Assignment statement, whereas the second figure below shows a logical Variable Declaration and Assignment statement.

Example of Numeric Variable Declaration and Assignment

Logical Variable Declaration and Assignment

How are Variables Assigned?

Look at the assignment statements a bit closer and notice that in both assignment statements, the name of the variable appears on the left side of the equal sign and assignment of the variable is on the right side. The variable name can never be on the right hand side of the assignment statement. At this time, do not be concerned with the content on the right hand side of the equations. The focus of this section is to make sure the syntax of the statement is completely understood.

Replace the word **VariableName** with a valid variable name, and replace the word assignment value with a valid expression.

Variable Assignment Syntax: VariableName = Assignment value;

In the example below, the variable CloseCompare is assigned a statement.

CloseCompare = Close > Close[1];

Referencing Past Values of Variables

In TradeStation, previous values for all variables, numeric and conditional, are saved for MaxBarsBack number of bars. These values can be referenced by users in their studies and systems.

Note: During the initial MaxBarsBack buffer, all value variables have a value of zero (0). So, be careful not to reference value variables for bars that fall into this initial buffer. When working with value variables, use the function CurrentBar, to make sure that this does not happen. CurrentBar tags the first bar following MaxBarsBack with the number 1. Each subsequent bar is given an ascending number such as 2, then 3, then 4, and so on.

The following is an example of a system that would not work because, on the first 10 bars, the value variables would not yet be set.

Incorrect:

Value1 = (Close + Open) / 2;
If Close > Value1[10] then Buy at market;

Correct:

Value1 = (Close + Open) / 2;
If CurrentBar >= 10 and Close > Value1[10] then Buy at market;

Correct:

If Close > (Close[10] + Open[10]) / 2 then Buy at market;

Note: The above example works because value variables are no longer being used. When using value variables, check to ensure that CurrentBar is equal to or greater than the study bar being referenced by the value variable.

What are Inputs and How are They Used?

Suppose a user wanted to write an Indicator that had one plot, which plotted the results of the Average function. The Average function needs two pieces of information before it can be calculated. First, it needs to know how many bars the Average is to be calculated on; second, it needs to know what to calculate it on, which is usually a price. Suppose a user wasn't sure about the length of the calculation, and that he or she wanted to work with it and see how it looked on the Chart without having to come back to the PowerEditor to make the change each time.

This is what an input is designed to do. An input is a substitute for a numeric or conditional expression (expressions were discussed earlier in this chapter). Inputs increase the flexibility of studies or systems, allowing a user to change the input values from the charting program when applying the study or system.

In the PowerEditor, inputs are declared and given default values. The default value is automatically used in the calculation of the Study or System until the input value is modified by the user in charting.

Inputs may be either conditional (i.e., they return a true or false result) or numeric. Acceptable expressions may be price or a function. The input default may not contain expressions that refer to either numeric variables or true or false variables.

Inputs must be declared and assigned a default value in the Input Declaration statement before the input is used in a rule. Inputs are not like variables; they cannot be assigned in an Assignment statement.

Declaring an Input

To declare an input, at the Input Declaration area (usually at the top of the code), type:

Input Declaration Syntax: Input: [Input Name]([Default value]);

The word Inputs can be substituted for the word Input in the syntax. The word Input Name would be replaced by a valid input name. Input names cannot exceed eight characters in length, cannot have spaces, nor can they have special characters in them (such as the characters above the numbers on a keyboard). Input names can have numbers in them; however, the number or numbers cannot start the input name. This means that all input names must start with a letter. In addition, input names cannot be one of the reserved words listed in the section Reserved Words. The words Default Value would be replaced with a number, usually 0, in the case of numeric inputs. In the case of logical inputs, the words Default Value would be replaced with either the word False or True.

If more than one input is declared, each should be separated by a comma. The list of inputs should be followed by a semicolon. The Input Declaration area may be as many lines as necessary, and the inputs may be listed in any order.

In the figures shown below, the input Factor is used so that the numeric value of Factor can be changed from within the Charting program. The default value is shown in parenthesis. This Study example allows a band line to be drawn at a varying percentage from the close.

Example of Numeric Input Declaration

Example of Logical Input Declaration

Defining and Using IF-THEN Statements

The IF-THEN statement is perhaps the most important statement in Easy Language. This statement type allows the user to check a condition to see if it is true or false, and then to proceed depending on the outcome of the condition.

It is often used to form hybrids with other statement types, such as PRINT, PLOT, and BUY AND SELL. For example, **if a condition is true then buy at the close**.

IF-THEN Statement Syntax: If [conditional check] then [action];

It is the **action** portion of the syntax that is often another statement type. In the two figures below, an IF-THEN statement is combined with the VARIABLE DECLARATION and BUY/SELL statements.

Example of If-Then Variable Assignment

The **conditional check** in the above Indicator was whether the Close was greater than the previous Close, or if it was less than or equal to it. Depending on the outcome, the variable TwoClose is assigned differently.

Example of If-Then BUY-SELL statement

In this System, we've made the comparison of the previous and current Close a logical variable. If the logical variable is evaluated as **True**, then the BUY statement is executed. If it is evaluated as **False**, then the SELL statement is executed.

In this example, we explicitly ask if **CloseCompare = True**; however, we could have dropped the = **TRUE**, because it is assumed. When checking the variable on the false side, it must be written out just as in the above figure.

Long IF statements can be difficult to understand and debug if they are not working correctly. Because the results of complex IF-THEN statements, including those with multiple conditional checks and actions, or those that use the logical operators AND or OR (discussed earlier in this chapter) are sometimes difficult to predict, Logic Tables are a good way to visualize these results. Remember that the result of an IF-THEN statement, the action, relies on whether certain statements are True or False

Simple IF-THEN statements are very useful to the individual who wants to perform one action depending on the outcome of the conditional check. Sometimes, however, a user will want TradeStation to execute several statements only if a particular condition is true, and another set of statements if the condition is false. When trying to accomplish this type of task, switch from the simple IF-THEN to what is known as a BLOCK statement.

Other times, a user may need to perform one action if the conditional check returns a false result and another action if it returns a true result. When trying to accomplish this type of task, switch to an IF-THEN-ELSE statement.

Constructing a Block Statement

The BLOCK statement is so named because a block of statements is executed at a time. If the result determined by the IF clause occurs, THEN calculations BEGIN and continue until the END clause is reached.

The following System, shown in the figure below, calculates Condition1 and Condition2 only if the current Close is greater than the previous Close.

Example of a Block statement

Notice that a semicolon is not placed at the end of the first line. Each line within the block ends in a semicolon because it is a complete statement. It is not until the END clause is completed that a semicolon is placed, signifying the end of the block. Also notice that the lines following the BEGIN clause are indented slightly. Although this is not required, it is strongly recommended that the user incorporate it in his or her writing style as it makes the code much easier to read and analyze.

Another System compares the current close with the previous close and, if the current close is greater, then resets the values of Value1 and Value2, as shown in the figure below.

Example of a Block statement

Constructing an IF-THEN-ELSE Statement

The outcome of the conditional check is either true or false. If one action is desired when the condition is evaluated as true, and another action is desired when the condition is evaluated as false, the efficient way to accomplish this is to use an IF-THEN-ELSE statement. Using the System in the figure below, rewrite it using the IF-THEN-ELSE statement instead of two separate simple IF-THEN statements. The result is shown in the second figure below

Example of a Block statement, using an IF - THEN statement

Example of an If-Then-Else statement

The IF-THEN-ELSE can be expanded even further to what is called the IF-THEN-ELSE IF statement.

Constructing an IF-THEN-ELSE IF Statement

Suppose a user wanted to construct an Indicator that checked the Close against the previous Close. If the Close of the previous day is greater, perform one action; if it is less than, perform another action; and if it is equal to, perform yet another action. The figure below illustrates how this is done.

Example of an If-Then-Else-If statement

Constructing a Nested IF Statement

Nested statements occur when an IF-THEN-type statement is inside a BLOCK statement (described earlier in this chapter).

Nested IF statements help save calculation time by beginning calculations only when an occurrence happens. In addition, they allow the flexibility of the conventional IF-THEN-ELSE logic.

A nested or cascading IF statement is used when two or more conditions must be satisfied before an action is taken. In the following example, the SELL is only taken if the Open is less than the previous Open and the BUY is taken when the Open is greater than or equal to the previous Open.

In addition, there are two types of BUY and SELL orders that can be placed. If the Close is less than the previous Close, Market orders are placed. If the Close is greater than the previous Close, then a Closing order is placed. The figure below is an illustration of a nested If statement.

Example of a Nested If statement

What Are Plot Statements?

Studies are graphical representations of some calculation performed on a particular security. The result might be a red line overlaid on top of the data, a histogram, or just a dot on a certain bar of data. These Studies help to identify trends and patterns.

The PLOT statement is exclusively used in the PowerEditor and can only be used in Studies. Therefore, do not attempt to use the statement in a System. The PLOT command is followed by a number between one and four; for example, PLOT1. Notice that there are no spaces in the PLOT command and that PLOT1 is used, not PLOTONE.

The PLOT statement must be written in the correct syntax so that it can be verified and function properly. Once in the correct syntax, the PLOT statement can be a stand-alone statement or it may be combined with other statement types, such as If-Then.

Plot Statement Syntax: PLOT#(Plot value, "Plot name");

The # sign must be replaced with the numbers one through four. Plot value is what the user wishes to plot. For example, if plot value was replaced with the word TwoClose, this study would plot the result stored in the numeric variable. The plot name is used for identifying each plot. If, for example, a user had a study with four different plots, he or she might have a difficult time remembering what each plot was when looking at it. By naming each plot line, access will be established through the Chart Status window to see the exact values of the plots. In the Chart Status window, the plot values for each bar will be under a column labeled with the plot name, as shown in the figure below.

Example of a Plot statement

Defining and Using Buy and Sell Statements

The BUY statement tells TradeStation to take a long position buying X number of shares/contracts, whereas the SELL statement tells TradeStation to take a short position selling X number of shares/contracts. The user specifies the number of shares/contracts that are to be bought or sold. When a BUY or SELL statement is executed, an open position is obtained.

At any given time, a user can have one of three possible positions. These can be either long, short, or flat. When a BUY statement is executed, the user assumes a long position. When a SELL statement is executed, a short position is assumed. Further on in the chapter, EXIT statements will be explored. These can take users from either a long or short position and return them to a **flat** position. **Flat** can also be considered as no position.

Some Systems have both BUY and SELL statements in them. Others have either a BUY or SELL statement and an EXIT statement. Some even have both a BUY and SELL statement along with two EXIT statements. Systems with just BUY and SELL statements are sometimes called reversal Systems.

BUY and SELL statements can only be written inside a system; therefore, they cannot be used in Studies or Functions. In addition, BUY and SELL statements can only be used in the PowerEditor, not the standard Editor.

Buy Statement Syntax: Buy [("Signal name")] [Number of Contracts][When] [Extra measure] [Order type];

Sell Statement Syntax: Sell [("Signal name")] [Number of Contracts][When] [Extra measure] [Order type];

Specifying When to Place the Order with a Buy or Sell Statement

The word When is an optional portion of the syntax. It is placed in the syntax to make the statement easier to read; however, it may be eliminated if desired. The order type dictates when the order is to be placed. If a user decides to use When in the syntax, it will usually be defined with words like tomorrow or today. The signal name syntax would only be applicable if a System had more than one BUY or SELL statement.

Specifying a Signal Name for a Multiple BUY and SELL System

A System may contain more than one BUY or SELL statement within it. When a System contains more than one signal, it is usually important to know which BUY or SELL statement got the user into the market and which signal generated the active order(s) on the screen. To do this, TradeStation allows users to specify a signal name, which can contain up to 25 characters. The signal name appears in parentheses immediately after the words BUY or SELL. Signal names are optional. If not using a signal name, do not include that portion of the syntax in the statement.

The name itself is surrounded by quotation marks, for example:

Buy("Signal1")

Sell("Signal2").....

Each signal name within a system must be unique; that is, no two signals can have the same name. The following examples, shown in the figure below, use signal names that, in the charting program, allow users to know which signal got them into the trade and which signal the active order(s) are for.

Examples of signal names

Specifying the Number of Contracts/Shares to Buy or Sell

In TradeStation, if the number of contracts/shares to buy or sell are not specified within the rules of a user's system, the default setting is assumed. (The default is set in the Charting program. From the Analysis menu, click on System, select the system, then click on the Costs Command Button.). In simple systems, this default may suffice. As users develop more sophisticated systems, they may want to vary the number of contracts/shares that are bought or sold, depending on the strength of the signal, current system performance, consecutive winning/losing streak, and other variables.

The figure below shows a system specifying the number of contracts/shares for the buy/sell signals.

Example of a system specifying Shares/Contracts

The system in the figure below uses a variable to specify the number of contracts/shares to trade based on the profit of the signal and the margin requirement per contract.

Example of a system using a variable to specify Shares/Contracts

Specifying the Type of Order with a Buy or Sell Statement

There are four basic types of orders in TradeStation: CLOSING, MARKET, STOP, and LIMIT orders (described in the sections below). A Closing order is placed and filled on the close of the current trading bar. Market orders are placed and filled at the open of the next bar. Stop and Limit orders are both placed at the open of the next bar; however, the orders are not filled until the Extra measure in the syntax is met at some point within the next bar. The Extra measure portion of the syntax is only used with Stop or Limit order types.

Two other types of trading orders that can be simulated use a combination of BUY or SELL and IF-THEN statements. These two order types are STOP CLOSE ONLY orders and STOP- LIMIT orders.

Writing a BUY or SELL Statement Using a Closing Order

Closing orders are filled on the current trading bar at the closing price of the bar. Examples of how they are written are included below. The figure below shows a simple closing order, which will generate a long position on the close of every bar. No conditions need be met in order for the BUY statement to be executed.

Example of a BUY statement using a simple closing order

The figure below shows a hybrid statement combining an IF-THEN statement with the same BUY statement used in the above figure.

Note: The text inside the curly brackets { } are comments used to point out the different parts of the statements. Text inside comment braces are not executed.

Example of a HYBRID statement using a BUY, SELL and IF-THEN statement (closing order)

The second System will generate the buy order when the condition is true or the sell order when the condition is false. In the figure below, the system has been applied to some data, and it is possible to determine where and when the orders were filled. The system was set with a special condition that would only allow one position at a time. This means that, when long, the system was not allowed to go long a second time.

Chart showing the results of the system shown in the previous figure

Writing a BUY or SELL Statement Using a Closing Order

Market orders are placed and filled at the open of the next bar. In a majority of Systems, a condition must occur before the order is placed. The figure below shows a hybrid statement combining an IF-THEN statement with a BUY statement.

Example of a HYBRID statement using a BUY, SELL and IF-THEN statement (closing order)

The figure below assumes the same conditions used above, except this time a market order will be used instead of a closing order.

Example of a HYBRID statement using a BUY, SELL and IF-THEN statement (market order)

The figure below examines the results and compares them to those of the closing order, noting

any differences in when the orders were filled.

Chart showing the results of the system shown in the previous figure

Writing a BUY or SELL Statement Using a Market Order

Market orders are placed and filled at the open of the next bar. In a majority of Systems, a condition must occur before the order is placed. For example, let's begin with the following condition:

Example of a HYBRID statement using a BUY, SELL and IF-THEN statement (closing order)

The figure below assumes the same condition used in the above figure, except this time a market order will be used instead of a closing order.

Example of a HYBRID statement using a BUY, SELL and IF-THEN statement (market order)

The figure below examines the results and compares them to those of the closing order, noting

any differences in when the orders were filled.

Chart showing the results of the system shown in the previous figure

Writing a BUY or SELL Statement Using Stop and Limit Orders

Stop or Limit orders are placed on the next bar and are filled when the Extra measure (see syntax) criterion in the syntax of the statement occurs.

A Stop order in a Buy statement tells TradeStation to take a long position on the next bar at the Extra measure price level or anything higher. A Limit order in a BUY statement tells TradeStation to take a short position on the next bar at the Extra measure price level or anything lower.

TradeStation normally fills an order at the value indicated by the stop price. Sometimes, however, the market might open above a Buy Stop price or below a Sell Stop price. In such a situation, TradeStation would automatically assume the order was filled at the open price. This reflects what would happen in a real-life trading situation.

A Stop order (BUY) may be pictured in this way,

... and a Limit order (BUY) in this way,

A Stop order in a Sell statement tells TradeStation to take a short position on the next bar at the Extra measure price level or anything lower. A LIMIT order in a SELL statement tells TradeStation to take a short position on the next bar at the Extra measure price level or anything higher.

A Stop order (SELL) may be pictured in this way,

... and a Limit order (SELL) in this way,

Let's begin with the condition in the following figure.

Example of a HYBRID statement using a BUY, SELL and IF-THEN statement (closing order)

Now use a Stop and a Limit order instead of a Market order. In the BUY statement, the Extra measure makes reference to the price Close. This refers to the Close of the bar on which the condition was met, not the bar on which the order is placed. The SELL statement explicitly references the Open of the next bar. If the of next bar was not included, it would have used the open of the bar on which the condition was met instead of the bar on which the order was placed. The two figures below illustrate this example.

Note: Only the Open, date, and time of the next bar can be referenced.
--

Example of stop and limit orders

Chart showing the results of the system shown in the previous figure

Writing a BUY or SELL Statement Using a Stop-Limit Order

A Stop-Limit order is used to be sure that the fill price is greater than a certain value but less than another value. In TradeStation, there is no direct way to place this order. Therefore, it must be placed as several hybrid statements using an IF-THEN-ELSE statement together with Stop and Limit orders.

The necessary code for accomplishing this task is displayed in the figure below. This code may be used as a shell and can replace the values stored in the variables ONBStopValue and ONBLimitValue with a user's own stop and limit values.

Example of BUY statement with stop-limit order

Writing a BUY or SELL Statement Using a Stop-Close-Only Order

Although in TradeStation, there is no direct way to place a Stop-Close-Only order, it can still be done. The order must be placed to BUY at the close of the current trading bar if the close is equal

to or higher than a certain price; likewise, the order must be placed to Sell on the close of the current trading bar if the close is equal to or less than a certain price. In TradeStation, it can be done by placing a hybrid statement using an IF-THEN statement together with a Closing order. In the figure below, an order is placed on the close at Open + 2 points or higher. This essentially accomplishes a Stop-Close-Only order.

BUY statement using a stop-close order

Exiting the Market

The EXITLONG and EXITSHORT statements are used to close out or liquidate long or short positions. The EXITLONG statement is used to close out a long position and the EXITSHORT statement is used to close out a short position. If, for example, a user wrote a System with one BUY statement and one EXITLONG statement, the execution of the BUY statement would take him or her **long**, while the execution of the EXITLONG statement would bring one back to a **flat** position.

The System examples that have been considered so far have dealt with EXITLONG and EXITSHORT statements and much of the syntax concerning their use may already be familiar. For example, SELL statements don't merely liquidate a current position, they reverse it.

Many times, however, a user will want to partially liquidate a current position or liquidate the position and remain flat. This is the purpose of EXITLONG and EXITSHORT orders.

ExitLong Statement Syntax: EXITLONG [("Signal name")] [number of contracts] contracts [from Entry("Entry name")] [when][extra measure][order type];

ExitShort Statement Syntax: EXITSHORT [("Signal name")] [number of contracts] contracts [from Entry("Entry name")] [when][extra measure][order type];

Specifying a Signal Name for Multiple EXITLONG and EXITSHORT Systems

A System may contain more than one EXITLONG and EXITSHORT statement within it. When this is the case, it is usually important for users to know which EXITLONG or EXITSHORT statement got them out of the market and which statement generated the active order(s) on the screen. To do this, TradeStation allows the user to specify a signal name containing up to 25 characters. The signal name appears in parentheses immediately following the words EXITLONG or EXITSHORT. Signal names are optional and only need to be used if there is more than one EXITLONG or EXITSHORT statement.

The name itself is surrounded by quotation marks; for example:

ExitLong("Signal1")

ExitShort("Signal2").....

Each signal name within a system must be unique that is, no two signals can have the same name. The following examples use signal names that, in the Charting program, will identify which signal got the user into the trade and which signal the active order(s) are for. The figure below illustrates an EXITSHORT signal called "Trgt1" and an EXITLONG signal called "Trgt2."

Example of signal names in an exit statement

Specifying the Number of Contracts/Shares to Exit

Unlike Buy and Sell orders, which can be hit many times (assuming that the setting that allows multiple entries has been turned on), an exit order, EXITLONG or EXITSHORT, can be hit only once for each instance of an entry.

When specifying the number of contracts, replace the words number of contracts in the syntax of the statement with a number. The number must be followed by the word contracts as seen in the syntax. Of course, if the number of contracts are not specified but the defaults are used instead, this section of the syntax [number of contracts] contracts is left out. When the number of contracts are specified by the user, the default values are overridden.

Consider the system in the figure below. If the buy is hit once and the EXITLONG is hit, a user would still be long nine contracts/shares. If the buy is hit more than once, the EXITLONG statement will exit one contract from each instance of the buy. Consequently, the user will still hold nine contracts/shares from each buy. Therefore, exit signals in TradeStation exit whatever number of contracts/shares specified from each instance of an entry signal.

Be certain that, if exit orders are used to liquidate a position entirely and remain flat, the EXITLONG and EXITSHORT statements cover all the potential BUY and SELL statements of the system. A system may contain as many exit signals as desired by the user.

Example of a system specifying share/contracts

Specifying From Which Entry to Exit

When there is more than one BUY statement, it becomes necessary for the user to specify from which buy signal to EXITLONG. The same can be said for the SELL and EXITSHORT statements. Another point at which users must specify the signal name from which they are exiting is when they are attempting to exit using a price anchored on the bar of entry. Figure 5-29 illustrates this concept.

Note: "AT\$" anchors the price relative to the bar on which the entry order was generated. This is explained further on in the chapter.

Example of a system specifying entry signals in the EXIT statement

Specifying When to Place the Order

In the BUY and SELL statements, the words **today** and **tomorrow** are optional; however, it is **strongly recommended** that these words be used to provide clarity to Systems. Stop and limit orders are always executed on the next bar, whereas closing orders are generated on the same bar

on which the condition is met. The reason these words are optional is because the order type automatically determines when the order is placed.

The word **When** is placed in the syntax to make the statement easier to read. If When is used in the syntax, it will usually be replaced as follows:

ExitLong at open of next bar;
ExitShort tomorrow at market;
ExitLong today on the close;
ExitShort tomorrow at high of today + 10 points stop;

Therefore:

ExitShort tomorrow at high of today + 10 points stop;

... is the same as ...

ExitShort at high + 10 points stop;

Specifying the Type of Order in an EXITLONG or EXITSHORT

There are four basic types of orders in TradeStation: CLOSING, MARKET, STOP and LIMIT orders (described in the sections below). A closing order is on the close of the current trading bar. Market orders are placed and filled at the open of the next bar. Stop and limit orders are both placed at the open of the next bar

; however, the orders are not filled until the Extra measure in the syntax is met at some point within the next bar. The Extra measure portion of the syntax is only used with stop or limit orders.

Exit orders without a stop or limit are always executed on the close of the current bar.

Therefore:

ExitLong today on the close;

... is the same as ...

ExitLong;

Exit at Market orders are always executed on the open of the next bar. Therefore:

ExitShort tomorrow at market;

... is the same as ...

ExitShort at market;

... or ...

ExitShort tomorrow at open of tomorrow;

... or ...

ExitShort at open of next bar;

Writing studies using this expanded style will help users to keep clear as to when the system is going to EXITLONG or EXITSHORT.

Writing Exit Orders That Encompass the Prices on the Bar of Entry

In trading, there are many times when a user may wish to anchor an exit signal to a specific price relative to the day the order was generated. To do this, it is necessary to use the AT\$ feature.

The syntax is: AT\$ expression

Using AT\$ before a mathematical expression anchors all references of price, function or variable to the bar on which the order was generated.

If AT\$ is not used, the expression refers to the current price. If the entry signal was hit five times, and if the AT\$ exit is hit, it will exit at the price of the bar on which each entry order was generated.

Obviously, AT\$ can only be used in an exit signal and may not be used in the same signal with the word Total.

The system in the figure below buys four contracts/shares at a time, but exits only two contracts/shares at a time. After the exit is hit once, it cannot be used again by this instance of the entry, so another exit must be written. A protective stop is also included here because the limits

may never be reached and the market may begin to turn against the trader. The price of the protective stop is determined by the close of the bar on which the order was generated.

Example of an exit statement

Writing an EXITLONG or EXITSHORT Using a Closing Order

Closing orders are filled on the current trading bar at the closing price of the bar. Let us examine how a closing order is written. The first figure below shows a simple closing order, which will liquidate a long position on the close. There are no conditions that have to be met in order for the EXITLONG statement to be executed. The second figure below shows a hybrid statement combining an IF-THEN statement with an EXITLONG and EXITSHORT statement.

Note: The text inside the { } curly braces are comments used to point out the different parts of the statements. Text inside comment braces are not executed.

Example of an EXITLONG statement using a simple closing order

Example of a hybrid statement using an EXITLONG, EXITSHORT and IF-THEN statement (closing order)

The system in the above figure will only generate the EXITLONG statement when the condition is true; furthermore, it will only generate the EXITSHORT statement when the condition is false. In the figure below, the system has been applied to some data; now let us see where and when the

orders were liquidated. The system was set with a special condition that would only allow one position at a time. This means that when long, the system was not allowed to go long a second time.

Chart showing the results of the system shown in the previous figure

Writing an EXITLONG or EXITSHORT Using a Market Order

Market orders are placed and filled at the open of the next bar. In a majority of systems, a condition must occur before the order is placed. Assume the following conditions:

Example of a hybrid statement using an EXITLONG, EXITSHORT and IF-THEN statement (closing order)

In the figure below, we use the same condition as in the figure above, except this time, a market

order instead of a closing order is used.

Example of a hybrid statement using an EXITLONG, EXITSHORT and IF-THEN statement (market order)

In the figure below, let us examine the results and compare them to those of the Closing order, looking for the differences in when the orders were filled.

Chart showing the results of the system in the previous figure

Writing an EXITLONG or EXITSHORT Using Stop and Limit Orders

A stop-limit order is used to be sure that the exit price is greater than a certain value but less than another value. In TradeStation, there is no direct way to place this order and, therefore, it must be placed as several hybrid statements using an IF-THEN-ELSE statement together with a stop and limit order. The necessary code for accomplishing this task is shown in Figure 5-38. This code may be used as a shell, and the values stored in the variables ONBStopValue and ONBLimitValue may be replaced with a user's own stop and limit values. This code is for EXITSHORT statements, practice writing it for an EXITLONG.

Example of an exit statement using a stop-limit order

Writing an EXITLONG or EXITSHORT Statement Using a Stop-Limit Order

In TradeStation, there is no direct way to place a stop-close-only order; however, there is a way to simulate it. The order must be placed to EXITLONG at the close of the current trading bar if the close is equal to or less than a certain price; likewise, to EXITSHORT on the close of the current trading bar if the close is equal to or greater than a certain price. In TradeStation, it can be done by placing a hybrid statement using an IF-THEN statement together with a closing order. In the figure below, a short exit is placed on the close at Open + 2 points or higher. This basically accomplishes a stop-close-only order. Practice writing the EXITLONG statement.

Example of an EXITSHORT statement using a stop-close-only order

Writing EXITLONG or EXITSHORT Statement Using a Stop-Close- Only Order

The PRINT statement is used to send specified information to a file, to the printer, or to the Print Log. The Print Log is a window in the charting module that stores the results of executed print statements. For more information on Print Logs, refer to Appendix D, Print Log.

Once in a file, on a piece of paper, or in the Print Log, the information is very useful for debugging Studies and Systems that are not working correctly. For example, assume a study that was supposed to return a value representing 10% of the bar's range was instead returning an incorrect result. This problem could be fixed by printing out the values for each bar and the price it on which it was based. The calculation could then be done by hand to see where it might be breaking down.

Print commands can be written anywhere in studies, systems, or functions. Like any other command, the PRINT statement must be written in the correct syntax so that it can be verified and can function properly. Once in the correct syntax, the PRINT statement can become a stand-alone statement or it may be combined with other statement types such as IF-THEN statements.

Print Statement Syntax: PRINT("Identifying text", print information[formatting]);

To understand the value of the PRINT statement in debugging signals, let us examine a relatively simple study with a very large problem as displayed in the figure below. The PRINT statement has some formatting in it that will be discussed in the Questions and Answers section.

Example of a PRINT statement

Is the problem clear? Look at Value1 again. If it is assumed that the close was 245.00 and the low was 240.00, what is Value1? $245.00 - 240.00 * 2 = ?$

The answer is not 10. Remember, multiplication is higher in precedence than subtraction; therefore, $245.00 - (240.00 * 2) = 245.00 - 480.00 = -235.00$.

When the calculations are run, obviously the trades generated are not the ones expected. Here is where a PRINT statement can help. Use the PRINT statement to provide a bar-by-bar display of VALUE1.

In a Print Log window, shown in the figure below, the PRINT statement displays information

similar to the following as each bar is completed.

Print Log window

Clearly, all those negative values indicate that something is wrong with VALUE1. The way it is written, VALUE1 equals $\text{Close} - \text{Low} * 2$, when it should equal $(\text{Close} - \text{Low}) * 2$. This would yield the correct result:

$$(245.00 - 240.00) = 5.00 * 2 = 10.00.$$

Using Print is an effective way to debug even complex signals. In fact, in very lengthy or complex signals, it is often advisable to break the signal into components. First, verify the component. Second, use Print to check that the results are those that are expected.

Using Print Statements

Previously when we thought of a variable, we associated a certain name with a value. The value would vary, with previous values being replaced by the most recent value, but at any time, there was only one value assigned to the variable. In some applications, it might be necessary to save values as they are received. Array variables allow a user to do this.

An array is a special type of variable. There are two major differences between array and non-array type variables:

1. Arrays need their own declaration line, whereas variables of type numeric, truefalse and string can be declared in the same VAR line.
2. Arrays can store from 1 to 16,000 elements at any one time. Numeric, truefalse and string type variables can only store 1 element at a time.

Array Declaration Statement: `ARRAY : MyArray[x](i);`

... where:

1. x refers to the number of elements in the array, and i refers to the numeric, string or truefalse value to which those elements will be initialized.
2. x and i must be written as real numbers. Inputs and variables cannot be used to replace x or i.
3. Each element is referenced via an index number between 0 and x.
4. You will get an Out of bounds error if you attempt to reference an index number less than 0 or greater than x.

Example 1:

```
INPUTS: Price(Close), Length(10);  
VARs: x(0), y(0), TgtNum(0), EntDay(), ValHit(FALSE);  
ARRAYS: TgtArray[100](0), DayArray[50](0), TFArray[75](FALSE);
```

TgtArray is a variable that stores 101 elements, and whose elements are all initialized to 0 (zero). Why 101? An array's first element is always indexed at 0; thus, TgtArray can store 0 to 100, or 101 numeric-type elements. DayArray can store 51 string-type elements, and TFArray can store 75 truefalse-type elements.

Example 2:

```
x = 1;  
y = x;  
IF DayOfWeek(Date) = x THEN DayArray[0] = Monday;  
IF DayArray[y-1] <> THEN DayArray[y] = DayArray[y-1];
```

In the first two lines of Example 2, the numeric variables x and y have been set to 1. The third line states that if the DayOfWeek function for date Date is equal to x (in this case number 1), then store the string Monday in DayArray element number 0. The fourth line states that if DayArray element y - 1 (in this case 0) does not contain a blank string, then copy the string value in DayArray element number 0 into DayArray element number 1.

Arrays can also be tied to a data series; for example:

```
Array : MyArray[20] (0,Data2);
```

...which represents a 21-element array (0-20), initialized to zero and containing information for the Data2 price series.

What are Arrays and How are They Used?

Loop statements automate repetitive calculation operations that are to be repeated, either for a set number of times (the FOR loop) or for so long as an event occurs or does not occur (the WHILE loop).

The FOR loop is predictable, because the user sets the number of times that it will repeat. The WHILE loop is not predictable, because it will be executed so long as the indicated condition is true.

TradeStation users tend to write a wide variety of signals, from simple to complex. Signals may be only a few words long, or they may be pages in length. The following statements suggest, to more advanced users, that signals of virtually unlimited complexity may be written successfully using the tools available in TradeStation.

The FOR Loop

The FOR statement instructs that a set of operations will be executed repeatedly while a progression of values is assigned to a value variable. The progression of variables can be ascending (TO) or descending (DOWNT0) the final value.

Example #1:

```
For Value1 = 0 TO 10 BEGIN  
  ^      ^      ^  
  1      2      3  
END;
```

Example #2:

```
For Value1 = 10 DOWNT0 0 BEGIN  
  ^      ^      ^  
  1      2      3  
END;
```

KEY: 1 = control variable, 2 = initial value, 3 = final value

If the initial value is greater than the final value when using the TO clause, or if the initial value is less than the final value when using the DOWNT0 clause, then the entire set of instructions is not executed at all.

Upon completion of the FOR statement, the control variable equals the final value.

The FOR statement is very powerful because it allows a set of instructions to be repeated for a predetermined number of times within a signal. This repetition would be essential if, for instance, a signal was written that used the second highest high of the last 10 bars.

The figure below contains an example that calculates a 10-bar moving average of the closing

price.

Example of a FOR loop

The WHILE Loop

Another type of loop statement is the WHILE statement. Unlike the FOR loop that performs a predetermined number of passes, the WHILE loop continues to loop only while a specified condition is true.

The format for the WHILE statement is the following:

```
WHILE <condition> BEGIN
any valid statements can go here
END;
```

The figure below uses the same example as above but writes it as a WHILE loop.

Example of a WHILE loop

The figure below finds the bar number of the last up bar of the last 50 bars.

Second example of a WHILE loop

In the above figure, the bar number of the last up bar is contained in the variable VALUE1. That is because an escape from a WHILE loop is always necessary so that passes do not continue endlessly. In this case, when Value1 >= 50, the loop stops.

What are Loop Statements and How are They Used?

In TradeStation, systems are limited to 64K of code in order to be verified. To break this limitation of system size, the user may include or "nest" systems within one another.

The included systems must be complete systems, not just code without buy and sell statements. An unlimited number of systems can be included in a system. If the system that is being created is becoming too large, it can be divided into smaller systems and then added together to create a single system.

There are several advantages to breaking up systems. The overall size can be greater than 64K as long as no one system is greater than 64K. A user may have a certain technique that he or she uses over and over, such as an exit technique. This technique can be written as a system and included in all systems.

In concept, there are two types of systems, one that references the open of the next bar and one that does not. Any system that references the open of the next bar should be an independent system included in the overall system. TradeStation prohibits user systems that are self-fulfilling. Therefore, a user cannot write a system that references tomorrow's open and buys today on the close.

If a system has Inputs, the primary system passes the input values to all secondary systems. Information cannot be passed back and forth between the systems. Each system has to be its own independent entity.

From the TradeStation PowerEditor, use the File - Open menu sequence to open a system that is currently written. Place your cursor where you want the system to be inserted.

Opening Bar Breakout system

Then, use the Tools - Function Wizard menu sequence to produce the Function Wizard. In the Function Wizard, click on System to Include in the Category window. Then, in the Function Wizard, locate in the Name window the system that is to be included in the system that is being written. Click on the system to select it. When the system is selected, click OK.

The selected system will be inserted at the point where the cursor was originally located, as shown in the figure below.

Sample Include system

This is an example that can be used with daily data. The first system is an entry system if there is a breakout of the opening price and the second system, which will include the first, exits on the close of the bar.

Write the first system and verify it. Write the second system using the Help Paste command. Select the Opening Bar Breakout system to have it included in the sample system. Verify the second system. Run the sample system on daily data in charting.

As shown in the two figures below, inputs can be added to make the system more flexible.

Open Bar Breakout system with inputs

Sample Include system

Include System Statements

In TradeStation, systems are limited to 64K of code in order to be verified. To break this limitation of system size, the user may include or "nest" systems within one another.

The included systems must be complete systems, not just code without buy and sell statements. An unlimited number of systems can be included in a system. If the system that is being created is becoming too large, it can be divided into smaller systems and then added together to create a single system.

There are several advantages to breaking up systems. The overall size can be greater than 64K as long as no one system is greater than 64K. A user may have a certain technique that he or she uses over and over, such as an exit technique. This technique can be written as a system and included in all systems.

In concept, there are two types of systems, one that references the open of the next bar and one that does not. Any system that references the open of the next bar should be an independent system included in the overall system. TradeStation prohibits user systems that are self-fulfilling. Therefore, a user cannot write a system that references tomorrow's open and buys today on the close.

If a system has Inputs, the primary system passes the input values to all secondary systems. Information cannot be passed back and forth between the systems. Each system has to be its own independent entity.

From the TradeStation PowerEditor, use the File - Open menu sequence to open a system that is currently written. Place your cursor where you want the system to be inserted.

Opening Bar Breakout system

Then, use the Tools - Function Wizard menu sequence to produce the Function Wizard. In the Function Wizard, click on System to Include in the Category window. Then, in the Function Wizard, locate in the Name window the system that is to be included in the system that is being written. Click on the system to select it. When the system is selected, click OK.

The selected system will be inserted at the point where the cursor was originally located, as shown in the figure below.

Sample Include system

This is an example that can be used with daily data. The first system is an entry system if there is a breakout of the opening price and the second system, which will include the first, exits on the close of the bar.

Write the first system and verify it. Write the second system using the Help Paste command. Select the Opening Bar Breakout system to have it included in the sample system. Verify the second system. Run the sample system on daily data in charting.

As shown in the two figures below, inputs can be added to make the system more flexible.

Open Bar Breakout system with inputs

Sample Include system

Working with Multiple Data Series Statements

Through the use of a multiple data Chart, TradeStation allows the user to reference up to 50 data files within a signal or study, e.g., data1, data2, data3....data50. Thus, different commodities, different time periods, or even fundamental outside data can be compared. For example, intramarket or intermarket comparisons can be made of IBM and the Dow Jones; February bellies and May bellies; or soybean and wheat.

When testing a system, always select the data needed to buy and sell as data1. Orders can be placed only for data1. For example, in comparing gold (data1) and silver (data2), the signal shown in the figure below would buy gold if it closes higher while silver closes lower:

Notice in this example that, where no data file is specified, data1 is assumed.

Note: When comparing data files of different time periods, set up the Multiple Data window so that the shortest time period is data1, the next shortest data2, etc. For example, data1 would be daily, data2 would be weekly, data3 would be monthly.

Numeric variables can be tied to a specific data series when using multiple data files. The default is data1 if a data series is not specified when declaring variables.

Special Variable Declaration Syntax: Vars : X1(0,Data1), X2(0,Data2);

Where **0** refers to the initialization value and **data1** (or data2) ties the variable to that data series and **X1** and **X2** are variables.

Also, the data series (or data number) to use when referencing simple functions or prices can be a variable.

Example:

Value1 = Average(Close,10) of data(X1);

...where **X1** is a variable whose value is between 1 and 50.

All predefined and non-predefined variables for which a data number is not specified automatically default to data1.

The resetting of variables and arrays for a data series is not resolved. Therefore, it is important to initialize conditional variables to "false" before using them, as they may have been used earlier in the signal.

What are the Components of Easy Language?

You can think of "components" as the building blocks of Easy Language. They are what nouns, verbs, adjectives, and prepositional phrases are to English.

The components of Easy Language include prices, inputs, variables, special characters, numbers, functions, basic rules, and special words.

What are Functions?

TradeStation is equipped with a tool called QuickEditor, which enables you to write your own indicators, ShowMe studies, PaintBar studies, simple trading systems and functions using Omega Research's Easy Language™. You can also use the PowerEditor, a separate Omega Research application, to write any type of analysis technique. When Omega Research developed Easy Language and the QuickEditor, they incorporated calculations and tasks that you would perform most often into your charting application for you as **Functions**.

These functions serve as building blocks for any study or system that you may want to write. They cannot be applied to data directly, instead, they must be 'called,' which means they must be used within a study or system. When you are designing a study or system, ask yourself: has or could any part of my study or system have been written as a function? If it has, when you are writing your study or system in QuickEditor, you can simply 'paste' the existing function within your study or system.

Over 150 functions have been incorporated into the QuickEditor and PowerEditor for you. And, Omega Research has made Easy Language flexible enough so that if, for some reason, you want to perform a task that is not already provided as a function, you can write your own function.

All functions automatically perform a specific task and return a numeric value, which may then be analyzed, used in an arithmetic expression, or used in another function.

A function may be an oscillator or an indicator commonly used in commodity or stock analysis, or it may be a simple time-saving calculation. A function usually requires some type of input, or **parameter**, which is placed within parentheses after the function name.

Parameters can be any valid numeric expression, including another function. In addition, functions may also contain Boolean operators or true/false inputs. The functions have been grouped into the following categories:

- Data Information
- Date & Time
- Drawing Objects - for use with the PowerEditor only
- Easy Language Functions
- Fundamental (Historical) Data
- Fundamental (Snapshot) Data
- Math & Trig
- Pager - for use with the PowerEditor only
- Performance Information - for use when creating a trading system
- Position Information - for use when creating a trading system
- String Related - for use with the PowerEditor only
- System Information (for plots)

System Information - for use when creating a trading system

Systems to Include - for use with the PowerEditor only

A detailed listing of each function can be found by referring to The Function Library.

Understanding Parameters

Most functions require parameters with which to perform their tasks. These parameters can be hard coded, that is given a specific value, or they can be given as variables, which means the value will vary depending on the value assigned to the variable. In addition, a function can be offset. This section discusses these concepts.

Most functions contain parameters, which are the values the function needs in order to perform its task, or calculate its formula. For example, one of the Study functions, Average, has two parameters: PRICE and LENGTH. The formula used in the Average function sums the prices associated with (n) number of bars together, and then divides this sum by the total number of bars summed.

With some parameters, you must choose which type of values to use. For example, with PRICE, you must specify what specific price to use. In this case, you could use the Close, Open, High, or Low price. However, you are not restricted to using such a simple value. In fact, you may choose to use a value that calls another function. For example, you could use the average of the RSI indicator values over the last ten bars as the value of the PRICE parameter.

The PRICE parameter in the Average function is said to be a **numeric series argument**. This is because it returns a number that has history. PRICE has history because a particular price value is tied to a specific bar used in the formula. The value of a numeric series argument varies depending on the specific bar used in the formula.

The LENGTH parameter in the Average function is said to be a **numeric simple argument**. This is because it returns a number that has no history. That is, the number of bars used to calculate the average remains constant. In most cases, you'll use a constant such as 5, 7, 25, etc. for the LENGTH. Once again, you aren't restricted to using such simple values, but whatever you replace a numeric simple argument with, it must remain constant from bar to bar.

Hard-Coding Parameters

When writing the study or system, you can write, or **hard-code**, the parameter values into the programming code, thus creating an inflexible value for the parameter. When you hard-code the parameters, that is, you write the values for the parameters into the programming code, you cannot change the value without opening the code for the analysis technique and changing that value throughout the code. Therefore, it is recommended that you use variables so that you have the ability to change the values of any parameters at will.

Using Variables

When writing a study or system, parameters don't necessarily have to maintain the same values. Instead, they can be replaced with **variables**. Variables are essentially place holders for the values, which you can replace at any time. Using variables results in two important benefits:

1. The value of a variable can be changed at will without having to change the master code stored in the library of the analysis technique. If hard-coded parameters are used, you cannot change the value without opening the code and changing that value throughout. With variables, you can change the values without opening and modifying the code, simply by assigning a different value to the variable.

2. You have the ability to test a series of values and see how they affect the performance of a system; this is called system optimization. Since system optimization is only performed on system variables and stops, the parameters cannot be hard-coded. In order to perform system optimization, you must use variables.

Omega Research recommends that you use variables as opposed to hard-coding the parameters so that you have the ability to change the values of all parameters at will.

Using Offsets

Most functions in the Study category can be offset, which means being able to return the value of a function from a previous bar. For example, if you want to find the average price of seven bars, 10 bars ago, the function would be written as follows:

Average(Close, 7)[10]

... where PRICE = Close, LENGTH = 7, and the offset = 10. This means that the function will go back 10 bars from the current bar, sum the Close prices of the seven previous bars and divide the resulting sum by seven.

You can also use a function on different price series, or data series. For example, suppose you are working with a chart containing more than one price series, such as the S&P500 cash as the first price series (Data1), and the S&P500 December futures contract as the second series (Data2). If you wanted to find the average from 10 bars ago for the two different price series, you would use the Average function with the following parameters and offset:

To apply the example to the S&P50 cash chart:

Average(Close of Data1, 7)[10]

To apply the same example to the December S&P futures contract chart:

Average(Close of Data2, 7)[10]

Notice that each parameter in the function is separated by a comma, and that the parameters are enclosed in parenthesis. Also, the offset is in square brackets and follows the closed parameter parenthesis. The offset value must be a constant number.

What are Prices?

In a real-time environment, there is activity between buyers and sellers. When a transaction is made, shares, in the case of stocks, or contracts, in the case of commodities, are exchanged. When this happens, the price at which that transaction took place is broadcast by the exchanges. The most common trading method is to build bars based on time intervals from these ticks. In most cases, a bar consists of four prices: open, high, low, and close. Generally, 5-, 30-, and 60-minute bars are some of the intervals used by clients.

Being able to reference these prices is necessary. In addition, being able to reference the date and times of these prices are also important skills. The words and abbreviations displayed below are all in the **Reserved Words List**.

A TradeStation user may refer to any of the price or price-related items shown below.

PRICE	ABBREVIATION	DESCRIPTION
DATE	D	The date of the bar
TIME	T	The ENDING time of the bar
OPEN	O	The first tick of the bar
HIGH	H	The highest tick of the bar
LOW	L	The lowest tick of the bar
CLOSE	C	The last tick of the bar
TICKS		Number of trades during a bar
UPTICK		Number of ticks greater than the previous tick during a bar (applicable only when testing intraday data)
DOWNTICK		Number of ticks less than the previous tick during a bar (applicable only when testing intraday data)
OPENINT (daily data only)	I	Number of contracts outstanding at the close of a bar
VOLUME only)	V	Number of trades occurring during the bar (daily data

What are Skip Words?

Easy Language contains a number of optional words, called skip words, that may be included in any statement to make it easier to read. Skip words offer a great deal of flexibility in the way statements are worded. When a signal is verified (compiled), skip words are ignored. Their use is a matter of personal preference. All skip words are also considered reserved words.

The list below includes the skip words that are permitted in Easy Language:

A	DOES	PLACE
AN	FROM	THAN
AT	IS	THE
BASED	OF	WAS
BY	ON	

What are Numbers?

In addition to words, Easy Language also recognizes numbers that are typed as numerals (i.e., 4, not four). A number may contain up to eight digits, an optional leading sign, and a single decimal point. Examples of numbers are shown below:

36 +452.0 -3621.5
89421.31 12345.678 0.23465

A number may not contain a comma, dollar sign, or space. In addition, a number may not contain a trailing sign or a space after a leading sign. The examples below show incorrect numbers followed by their correct versions.

Incorrect	Correct
2,100	2100
\$300.26	300.26
300+	300
- 200	-200
256-	-256

Identifying Prices on a Desired Bar

Bars in the past can be referenced in one of two ways. The first is to explicitly type out the words:

bars ago

The second is a method that will help the user to accomplish the same task more concisely. The shortcut is to place the number of bars being referenced back within square brackets [#].

The current trading bar is defined as the most recent bar of data. If the bar being referenced is the current bar, the user may choose to use 0 bars ago or [0]; however, this is not necessary. If the item being referenced has no reference information, it is assumed that the current bar is being referenced.

The maximum number of bars that can be referenced back from the last completed bar (referred to in TradeStation as the current bar) currently is 999.

Long Method	Short Method	Description
0 Bars Ago	[0]	The current trading bar
1 Bar Ago	[1]	One bar prior to the current trading bar
2 Bars Ago	[2]	Two bars prior to the current trading bar
3 Bars Ago	[3]	Three bars prior to the current trading bar

Frequently, users will want to reference the prices from a particular bar. Below is an example of how this may be put into action. The examples demonstrate that the same statement can be written in a number of different ways.

OPEN OF 2 BARS AGO
OPEN[2]
O[2]

When describing a price, the description factor (open, high, low, etc.) must be written before the time factor (2 bars ago, 3 bars ago, etc.). Never use "yesterday's high" or "today's close," as Easy Language does not recognize the possessive syntax.

What are Points?

In commodity trading, a point is the minimum interval by which trades in that commodity are measured. Thus, if a market trades in 8ths, as in grains or stocks, a shift of one point represents a move of 1/8. Bonds trade in 32nds; a one-point move in bonds equals 1/32nd. The S & P moves in 100ths; a one-point move in S & P equals 1/100.

When referring to a one-point move in soybeans or most stocks, for example, it is simpler to refer to one point than to write 0.125, the decimal equivalent of 1/8.

This is an important concept that must be completely understood.

If a market trades in 100ths, then 1 point = .01
If a market trades in 32nds, then 1 point = .03125
If a market trades in 8ths, then 1 point = .125

When a Study or System that uses the term "points" is applied (assume the term "3 points" is used), TradeStation automatically identifies the security to which the study or system is applied and divides the points the user specifies by the fraction the security trades in.

If the study is tested using bonds data, the three points specified will be interpreted as 3/32. With grains' data, the same reference to three points will automatically be interpreted as 3/8. With Pork Bellies (which trades in 100ths), a reference to three points will be interpreted as .03.

TradeStation knows what interval each commodity or stock is traded in and internally stores all prices in decimal format. Therefore, the word "point" or "points" converts the number before it to its correct decimal equivalent. If this study is being tested with grain data, three points will be converted to 3/8 and stored as .375. If the statement refers to bonds, for example, which trades in 32nds, then three points would equal 3/32nds, or .09375.

These are the three primary rules regarding points:

1. Generally, the only time the words "point" or "points" are used is when the user writes a specific number (for example, 3) in the Study and adds or subtracts that number from a price.
2. Because prices are already in decimal format, never use the term "points" when referring to the result of the subtraction of two prices. The following example is used to illustrate this incorrect usage:

Open - Close[1] points

3. Do not use the word point(s) when entering the amount in decimal format. For example if a user wanted to find the open plus one point, he or she could write this in one of two ways:

Open + .125

Open + 1 point

It could not be written as follows:

Open + .125 points

Creating Work That is Easy to Read

Comment braces allow users to write notes to themselves in their systems, studies, and functions. Information enclosed within comment braces is ignored by TradeStation. Personal notes can be written about the study, how it works, or anything else desired.

Always begin a comment area with the left brace and end it with the right brace{ }. Comment braces can be written in both the PowerEditor and the QuickEditor inside the charting module. The figure below shows an example of comment areas.

Example of comment areas

Collecting Sufficient Data to Perform All Calculations

When TradeStation performs historical calculations, it moves an imaginary pointer through a contract data file. On any given bar, it knows the prices of that bar as well as the prices up to **MaxBarsBack** bars ago. MaxBarsBack is a buffer that stores the number of bars that can be referenced in a trading signal. If a signal incorporates a moving average of 50 bars, for example, beginning 10 bars ago, MaxBarsBack must be set to at least 60. The MaxBarsBack buffer is filled so that calculations begin on the first day after the buffer is filled.

Market Increments and "Rounding"

TradeStation automatically knows the minimum increment that every market trades in, and the program rounds certain prices to reflect these minimum intervals. As a result of this attention to detail, all orders can be filled. With TradeStation, it is impossible to generate an order that cannot be filled because the price falls between minimum market increments.

For example, a user may be trading S&P, which moves in minimum .05 increments; however, his or her signal may read:

Buy at close of today + Value1 Stop;

If Value1 = 240.11, the order cannot be filled because the minimum S&P increment is .05. As a result, TradeStation will automatically round up the price to 240.15 so that the order can be filled.

If the market were T-Bonds, which moves in 1/32nd increments, prices would be rounded to the appropriate 32nd.

TradeStation automatically rounds prices in the following way:

Order Type Price Rounded Up or Down to Nearest Increment

Buy Stop	Up
Sell Stop	Down
Buy Limit	Down
Sell Limit	Up

Reserved Words

Every specialty language contains words that carry particular significance. For example, certain words that have common meanings in everyday communications also have special meaning in medical or legal terminology. Because their meaning is interpreted in specific ways, care must be taken when using them. In Easy Language, each of these words is known as a reserved word, indicating that TradeStation consistently attaches the same special definition to the word or phrase. When a reserved word is used, be certain that its special meaning is understood.

Reserved also implies that these words cannot be used to define anything other than their set definition. When naming inputs and functions, reserved words may not be used.

Easy Language recognizes the following words, phrases, and operators listed below as reserved words. Reserved words cannot be used for variable, array, input, or function names.

Word or Symbol	Definition
,	Comma, used in formulas
.	Period
:	Colon
;	Semi-colon
" "	Quotation marks
(	Open parenthesis, used in formulas
)	Close parenthesis, used in formulas
*	Multiplication sign
+	Addition sign
-	Subtraction sign
/	Division sign
<	Less than sign
<=	Less than or equal to sign
< >	Not equal to sign
=	Equal to sign
>	Greater than sign
>=	Greater than or equal to sign
[	Used in bar offsets. For example, [3] is equivalent to three bars ago. Also used to identify array elements. For example, MyArray[4] is element number four in the array called MyArray.
]	See above.
{	Called the open comment brace, this symbol marks the beginning of a comment area. Users may write personal notes inside the comment area. The information inside the comment brace is ignored by TradeStation. Must be accompanied by a close comment brace to be successful.
}	Called the close comment brace, this symbol marks the end of a comment area. Must be accompanied by an open comment brace to be successful.
#BEGINALERT	Initiates alert procedure when true.
#BEGINCMTRY	Initiates commentary procedure when true.
#BEGINCMTRYORALERT	Initiates alert or commentary procedure when true.
#END	Closes the alert or commentary procedure.
A	Skip word - used to clarify statements.
	Works together with the Key word "Crosses." Used to detect when one value exceeds another. For example: RSI crosses above 80.
ADDTOMOVIECHAIN	Assigns a movie file to a reference number.
AGO	Refers to the bar number being referenced. For example: Open of 3 bars ago.
ALERT	Triggers an audible alert on the close of the bar, except if Update every tick is turned ON. For example: If C > O then Alert = true;
ALERTENABLED	Checks to see if Enable Alert check box is checked.
ALL	Used in exit orders to control # of contracts.
AN	Skip word - used to clarify statements.
AND	Use to link multiple conditions. For example: If Condition 1 and RSI(C,14) < 80.....Both conditions must be true in order for the entire statement to be true.

ARRAY(S)	Allows a user to save values as they are received. Each array has a certain number of elements associated with it.
ARRAYSIZE**	
ARRAYSTARTADDR**	
AT\$	Used in exit signals to anchor the prices on the bar of entry. For example: AT\$ Low refers to the low of the bar on which the order was generated.
ATCOMMENTARYBAR	See CHECKCOMMENTARY.
BAR(S)	Used to refer to specific bars. For example: Low of 1
bar ago.	
BASED	Skip word - used to clarify statements.
BEGIN	Used together with other Key word such as If, THEN, END, and ENDIF to create Block If Statements, Block If/Else Statements, or Loop statements.
BELOW	Works together with the Key word "Crosses." Used to detect when one value exceeds another. For example: RSI crosses below 80.
BOOL**	
BUY	Enter a long position in the market.
BUYALERT*	
BY	Skip word - used to clarify statements.
BYTE**	
C	Short for Close.
CHAR**	
CHECKALERT	Checks the status of an alert condition on a bar-by-bar basis for real-time data, and on the last bar of historical data.
CHECKCOMMENTARY	Checks the status of commentary on a bar-by-bar basis for real-time data, and on the last bar of historical data.
CLOSE	Refers to closing price of a bar.
COMMENTARY	Prints commentary to commentary window.
COMMENTARYCL	Sends line return to commentary window.
COMMENTARYENABLED	Checks to see if Enable Commentary check box is
checked.	
CONDITION 0-99	True/False variables. Used to store the return of a true/false statement. For example: Condition1 = C > O....
CONTRACT(S)	Used with other Key words such as BUY, EXITLONG, EXITSHORT, and SELL. Refers to the number of future contracts purchased. For example: Buy 2 contractsalso shares.
CROSS(ES)	Works together with the Key words Above and Below.
D	Short for Date
DATA1-50	The data series a System, Study, or Function is referencing. For example: Open of Data2....
DATE	Format: YYMMDD. Refers to a stamp place on all bars of data. For example: If Date = 930705 then
DAY(S)	Used to refer to specific bars. For example: Low of 1 day ago. In the previous example, if daily data were being loaded, the word day would be interpreted in the traditional way. However, if intraday data were being loaded, it would mean 1 bar ago
DEFINEDLLFUNC**	
DOES	Skip word - used to clarify statements.
DOUBLE**	
DOWNTICKS	Sum of ticks within a bar that have a lower value than the previous tick or have a value equal to the previous tick.
DOWNTWO	One of the key words in a loop statement. If the initial value in the loop is less than the final value, use DOWNTWO.
DWORD**	
ELSE	The ELSE clause allows users to specify what should be done if the condition in the IF statement is false.
END	Must match begin...Defines termination of a complex
statement.	
ENTRY	Used in exit order to control scope of this order.
EXITLONG	Used to partially or completely liquidate a long position.
EXITSHORT	Used to partially or completely liquidate a short position.

FALSE	Used to set a condition variable to false or to check the status of the condition variable.
FILE specified file.	Used in a print statement to send the contents to a
FLOAT**	
FOR	An element of the Loop statement. The FOR statement instructs that a set of operations will be executed repeatedly while a progression of values is assigned to a value variable.
FROM	Skip word - used to clarify statements.
H	Short for High.
HIGH	The highest tick of the bar.
I	Short for OPENINT.
IF	Used to check the status of a condition or set of
conditions.	
INCLUDESYSTEM	Allows the user to include one system within another. Makes it possible to write larger systems.
INPUT(S)	Used as a substitute for either numeric or true/false values. Allows greater flexibility by enabling input values to be changed on the fly.
INT**	
IS	Skip word - used to clarify statements.
L	Short for Low.
LIMIT	Type of Buy/Sell order...this price or better.
LONG**	
LOW	Lowest tick of the bar.
LPBYTE**	
LPCHAR**	
LPDOUBLE**	
LPDWORD**	
LPFLOAT**	
LPINT**	
LPSTR**	
LPWORD**	
MAKENEWMOVIREF	Used to obtain or allocate a new movie reference to which to assign a movie file in order to play it.
MARKET	Refers to the OPEN of the next bar. Type of
order.	
MOC	Market on close type of order. Use Close order.
MULTIPLE**	
NEXT	Used with bar to refer to one bar in future Next Bar.
NOT*	
NUMERIC	Define input to function as a number.
NUMERICSERIES	Define input to function as a number with
MaxBarsBack history.	
NUMERICSIMPLE	Define input to function as a number without
history.	
O	Short for Open. Do not confuse with
0(zero).	
OF	Skip word - used to clarify statments.
ON	Skip word - used to clarify statements.
ONLY	Used in exit orders to control # of contracts.
OPEN	The first tick of the bar.
OPENINT	Number of contracts/shares outstanding at the close of a
bar (daily data only).	
OR	Used to link multiple conditions. For example: If Condition1 or RSI(C,14) < 80..... Either one of the conditions must be true in order for the entire statement to be true. Boolean logical operator.
OVER	Used with CROSSES.
PAUSE*	
PLACE	Skip word - used to clarify statements.
PLAYMOVIECHAIN	Plays the movie identified with the reference number and file assigned to movie chain.
PLAYSOUND	Plays a .WAV file.

PLOT
 PLOT1-4
 desired.
 POB*
 POINT(S)
 commodity are measured.
 PRINT
 variety of places.
 PRINTER
 the printer.
 PROFIT*
 PROTECTIVE*
 REPEAT*
 SCO*
 SCREEN*
 SELL
 SELLALERT
 SHARE(S)
 of shares desired.
 SKIP*
 STOP
 STRING
 STRINGSERIES
 inputs.
 STRINGSIMPLE
 inputs.
 T
 TARGET*
 TEXT
 string.
 THAN
 THE
 THEN

 THIS
 TICKS
 TIME
 TO

 TODAY
 TOMORROW
 TOOL_BLACK
 drawing object.
 TOOL_BLUE
 drawing object.
 TOOL_CYAN
 drawing object.
 TOOL_DARKBLUE
 drawing object.
 TOOL_DARKCYAN
 drawing object.
 TOOL_DARKGREEN
 drawing object.
 TOOL_DARKMAGENTA
 to a drawing object.
 TOOL_DARKRED
 drawing object.
 TOOL_DARKYELLOW
 drawing object.
 TOOL_DARKGRAY
 drawing object.
 TOOL_DASHED
 drawing object.

usingThePowerEditor.doc
 Followed by 1..4. Display value of study on chart.
 Used in indicators to plot whatever is

Minimum interval by which trades in that

Designed to provide the opportunity to print information to a

Used with a PRINT statement to direct output to

Stop-Close-Only

Takes the user into a short position.

Used in Buy and Sell statements to specify the quantity

Type of order - "or worse."

Type of string used for function inputs.

Type of string used for function

Type of string used for function

Short for Time.

Undocumented function returns character

Skip word - used to clarify statements.

Skip word - used to clarify statements.

The second part of an IF-THEN statement. The "IF" checks the condition, whereas "THEN" gives the action to be executed.

Skip word - used to clarify statements.

Sum of UpTicks and DownTicks.

The ENDING time of the bar.

One of the keywords in a FOR Loop statement. If the initial value in the loop is greater than the final value, use "TO."

Refers to the current bar.

Refers to the next bar.

Numerical constant used to assign a color to a

Numerical constant used to assign a color to a

Numerical constant used to assign a color to a

Numerical constant used to assign a color to a

Numerical constant used to assign a color to a

Numerical constant used to assign a color to a

Numerical constant used to assign a color

Numerical constant used to assign a color to a

Numerical constant used to assign a color to a

Numerical constant used to assign a color to a

Numerical constant used to assign a line style to a

TOOL_DASHED2 drawing object.	Numerical constant used to assign a line style to a
TOOL_DASHED3 drawing object.	Numerical constant used to assign a line style to a
TOOL_DOTTED drawing object.	Numerical constant used to assign a line style to a
TOOL_GREEN object.	Numerical constant used to assign a color to a drawing
TOOL_MAGENTA drawing object.	Numerical constant used to assign a color to a
TOOL_RED drawing object.	Numerical constant used to assign a color to a
TOOL_SOLID drawing object.	Numerical constant used to assign a line style to a
TOOL_WHITE drawing object.	Numerical constant used to assign a color to a
TOOL_YELLOW to a drawing object.	Numerical constant used to assign a color
TOTAL	Exits the amount specified on all open positions. For example, if the user goes long twice and each time acquires three contracts, and if he or she has an exit signal that reads If C > C[1] then ExitLong 5 contracts TOTAL on the close, the following would take place: The system would buy three contracts, then another three contracts. Once the IF condition was met, the user would be exited from the first three contracts, and then from two of the three contracts from the second buy.
TRUE	Used to set a condition variable to true or to check the status of the condition variable or Boolean expression.
TRUEFALSE	Defines variable or input to a function as a
Boolean value.	
TRUEFALSESERIES	Boolean input to function with MaxBarsBack
history.	
TRUEFALSESIMPLE	Boolean input to function with no history.
UNDER	Works together with the Key word "Crosses." Used to detect when one value exceeds another. For example: RSI crosses above 80.
UNSIGNED**	
UNTIL*	
UPTICKS	Number of ticks greater than or equal to the
previous tick during a bar.	
V	Short for Volume.
VALUE0-99	Pre-declared numeric variables.
VAR(S)	Short for Variable(s). Declaration statement.
VARIABLE(S)	Used to declare a user's own variables.
VARSIZE**	
VARSTARTADDR**	
VOLUME	Number of trades that occurred during the bar (daily,
weekly, monthly data only)	
WAS	Skip word - used to clarify statements.
WHILE	Key word of a looping structure; While <expression>
begin....end;	
WORD**	
YESTERDAY	Refers to the previous bar. Not necessarily a day.
* Indicates that words are reserved for compatibility. They are not functional in TradeStation at this time.	
** Indicates that these are Dynamic Link Library (DLL) words. See file called USERDLLS.WRI stored in the C:\OMEGA\PROG\ directory for further explanation.	

Note: Words may be typed using upper case or lower case letters, or any combination of the two. For example, HIGH and high mean the same thing.